
Universidade de Aveiro
2015

Departamento de Eletrónica, Telecomunicações e Informática

JOANA COELHO VIGÁRIO

UTILIZAÇÃO OTIMIZADA DE RECURSOS EM TESTES DE SOFTWARE NA CLOUD


Universidade de Aveiro
2015

Departamento de Eletrónica, Telecomunicações e Informática

JOANA COELHO VIGÁRIO

UTILIZAÇÃO OTIMIZADA DE RECURSOS EM TESTES DE SOFTWARE NA CLOUD

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor Cláudio Jorge Vieira Teixeira, Equiparado a Investigador Auxiliar e do Doutor Joaquim Manuel Henriques de Sousa Pinto, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho aos meus pais e irmã pelo apoio incansável durante todo o percurso académico e especialmente durante a concretização deste projeto final.

o júri

presidente

Prof. Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

Prof. Doutor Fernando Joaquim Lopes Moreira
Professor Associado do Departamento de Inovação, Ciência e Tecnologia da Universidade Portucalense

Doutor Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar da Universidade de Aveiro

agradecimentos

Quero agradecer aos meus pais e irmã por sempre me terem encorajado a ser melhor todos os dias, a ultrapassar os obstáculos e lutar pelos meus objetivos. Um obrigada nunca será suficiente para mostrar a gratidão que tenho para convosco.

Não obstante da importância neste percurso, aos meus avós, tios, e primos, obrigada.

Quero agradecer à Joana e à Rita que sempre estiveram presentes ao longo de todo o percurso, nos bons e maus momentos e me ajudaram a retirar as pedras do caminho quando as mesmas não permitiam avançar.

Aos professores Cláudio Teixeira e Joaquim Sousa Pinto que me confiaram este desafio, fazendo-me crescer a nível profissional.

Aos meus amigos, colegas de trabalho e academia, que me acompanharam nas diferentes fases de todo o percurso realizado até este momento, muito obrigada por me apoiarem independentemente de que circunstância fosse.

Com toda a sinceridade, muito obrigada a todos.

palavras-chave testes, *software*, arquitetura, c*loud*, integração, elasticidade, sistema de informação

resumo Hoje em dia os sistemas podem evoluir rapidamente, e a esse crescimento está associada, por exemplo, a adição de funcionalidades, ou mesmo a mudança de perspetiva do sistema, por requisito dos *stakeholders*. Estas condicionantes exigem o desenvolvimento de testes de *software* para a validação dos sistemas. Executar sequencialmente uma grande bateria de testes é impensável porque a execução pode levar horas. Contudo, os testes podem ser executados mais rapidamente num ambiente distribuído e com rápida disponibilização de sistemas previamente configurados, como é o caso de *Cloud Computing*. Cada vez mais se procura a automação para todo o processo, incluindo integração, compilação, execução de testes e gestão de recursos da *Cloud*.

Esta dissertação pretende demonstrar a aplicabilidade da prática de *Continuous Integration* (CI) em Sistemas de Informação, para automação da compilação e execução de testes de *software,* num ambiente distribuído de *Cloud Computing,* com o propósito de conseguir a otimização e elasticidade dos recursos disponibilizados pela mesma.

keywords tests, software, architecture, cloud, integration, elasticity, information systems

abstract Nowadays systems can evolve quickly, and to this growth is associated, for example, the production of new features, or even the change of system perspective, required by the stakeholders. These conditions require the development of software testing in order to validate the systems. Run a large battery of tests sequentially can take hours. However, tests can run faster in a distributed environment with rapid availability of pre-configured systems, such as Cloud Computing. There is increasing demand for automation of the entire process, including integration, build, running tests and management of Cloud resources.

This dissertation aims to demonstrate the applicability of the practice Continuous Integration (CI) in Information Systems, for automating the build and software testing performed in a distributed environment of Cloud Computing, in order to achieve optimization and elasticity of the resources provided by the cloud.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ACRÓNIMOS

AD	*Active Directory*
API	*Application Programming Interface*
CI	*Continuous Integration*
IIS	*Internet Information Services*
OACV	Ordem dos Advogados de Cabo Verde
SCV	*Sistema de Controlo de Versões*
SIJ	Sistema de Informação da Justiça de Cabo Verde
SIPP	Sistema de Informação do Processo Penal
SO	Sistema Operativo
TFS	*Team Foundation Server*
VM	*Virtual Machine*
VS	*Visual Studio*

1 INTRODUÇÃO

1.1 ENQUADRAMENTO

A conceção de um sistema de informação não se limita ao desenvolvimento de código. Existe a necessidade de entender a visão do cliente relativamente ao sistema solicitado, por análise de requisitos, e também de compreender a indústria onde o mesmo se destacará. O desenho de uma arquitetura funcional e o planeamento de tarefas e iterações do produto é igualmente necessário, para suporte às equipas de trabalho. A avaliação do produto por parte do cliente e potenciais utilizadores é bastante importante para *feedback* às equipas de desenvolvimento. Por fim, é essencial testar o sistema e todas as suas componentes, de modo a garantir que este está pronto para publicação no mercado e que se comporta como o cliente pretende. A prática de realização de testes de *software* é essencial e é uma mais-valia à conceção de um sistema de informação.

Hoje em dia, os sistemas podem evoluir rapidamente, e a esse crescimento está associada, por exemplo, a adição de funcionalidades, a mudança de perspetiva do sistema por requisito dos *stakeholders* [1], ou a constante mudança dos mercados, que forçam os sistemas a satisfazer apressadamente as necessidades dos clientes e utilizadores.

A exigência de resposta imediata às mudanças requer que sejam realizados testes de *software* que permitam garantir o correto funcionamento de uma aplicação, tanto na perspetiva de um todo, como de uma parte. Os testes de *software* permitem validar que os pequenos blocos de código integrados no código estável (como métodos ou funções) funcionam como esperado e de forma correta, e ao mesmo tempo validam que a nova integração não danificou todos os conteúdos previamente funcionais.

Esta dissertação tem como sistema base de estudo o Sistema de Informação de Justiça de Cabo-Verde (SIJ). O SIJ está em desenvolvimento desde 2009, está em produção

desde meados de 2014, e conta com uma bateria de mais de 3000 testes. Num universo de mais de 3000 testes é crucial ter uma arquitetura que ajude à preparação e execução dos testes no mínimo tempo possível. Devido à complexidade e vasta gama de testes do SIJ, outrora a execução dos mesmos demorava entre 10 a 14 horas [2]. A justificação para estes tempos elevados está relacionada com as condições do ambiente de execução dos testes: execução sequencial, o que era incomportável. Depois de um trabalho de análise e alteração de configuração de estrutura de testes, foi possível diminuir o tempo de execução para 2.5 a 3 horas [2], utilizando uma solução distribuída.

1.2 MOTIVAÇÃO

As pessoas envolvidas no desenvolvimento de um sistema de informação têm de responder às exigências de forma eficaz, independentemente se as mesmas desempenham papéis específicos (compondo equipas especializadas) ou papéis mais genéricos. As equipas de trabalho desempenham diferentes tarefas como analisar requisitos, desenhar casos de uso, desenvolver código, realizar testes, avaliar o produto ou gerir o projeto.

A equipa responsável pela realização de testes de *software* poderá fazer parte do projeto como um todo, porém deverá ser uma equipa externa ao desenvolvimento. Este facto está relacionado com a vantagem de ter uma visão abstrata sobre a solução, conseguindo ser mais crítico e explorativo sobre a vasta gama de possibilidades de testes a realizar. Normalmente esta equipa trabalha sob a pressão de garantir que todas as componentes são testadas com todas as combinações possíveis, no mínimo tempo possível, dentro dos limites impostos pela metodologia de trabalho seguida. Apesar da pressão que se possa sentir, a equipa deve elaborar os testes minuciosamente, conseguindo ao máximo replicar todas as interações realizadas no sistema [3].

A equipa de testes do SIJ trabalha sobre uma arquitetura que permite executar os testes de forma distribuída, num ambiente de *Cloud Computing*, em recursos virtualizados, como máquinas virtuais (VMs), apoiando-se num *deploy* automatizado para a configuração dos testes nas mesmas máquinas. A configuração é manual, estática e

gerida apenas pelo *tester*. Devido ao facto da configuração ser estática, os recursos da *cloud* estão sempre a ser consumidos, mesmo quando não estão a ser necessários para a execução dos testes. A estrutura possibilita que os testes possam ser executados em menos tempo, como mencionado anteriormente, permitindo que a equipa de testes consiga a execução em qualquer momento, com o objetivo de analisar os resultados e verificar o estado atual do sistema.

A execução de testes e os resultados da mesma são geridos pelo *tester*. Os resultados só são disponibilizados às entidades envolvidas, quando o *tester* decide realizar a execução. O facto do conhecimento da execução de testes ser detida pelo *tester*, e de ser o mesmo a divulgar os resultados, limita que a execução seja sempre realizada por ele ou pela equipa de testes. Isto impede que outras pessoas envolvidas no projeto possam executar os testes remotamente, para análise antecipada dos resultados, por falta de conhecimentos técnicos da estrutura atual.

Para as equipas de trabalho seria útil que a execução dos testes fosse um processo automático, em todas as fases que o completam, desde a configuração dos recursos virtualizados necessários para a execução remota, até à inicialização da execução e a recolha dos resultados, fazendo a gestão dinâmica da *cloud.*

1.3 OBJETIVOS

Dados os constrangimentos apresentados pretende-se, a partir da arquitetura atual, criar uma solução que permita rentabilizar a elasticidade da *cloud,* e que ao mesmo tempo, reduza o tempo de execução dos testes e facilite o trabalho de todas as equipas aquando da execução dos mesmos. Para a automação deste processo é sugerida a criação de um *script* que efetue todos os passos de configuração da execução de testes remotos. Com a comunicação com a *cloud* poderá ser possível a gestão e controlo dos recursos necessários, dando liberdade à equipa de testes de definir quantos recursos serão utilizados para cada execução, controlando a velocidade de execução dos testes remotos na estrutura da *cloud*, de modo a diminuir os tempos de execução dos mesmos.

A aplicação da nova arquitetura poderá rentabilizar e otimizar a distribuição dos recursos da *cloud*, dependendo das necessidades de cada utilizador da mesma, diminuindo a sobrecarga do *provider* de *Cloud Computing* em questão. Os recursos poderão ser consumidos de forma otimizada, pelos utilizadores diurnos da *cloud*, se as VMs de suporte à execução de testes remotos não estiverem a sobrecarregar a *cloud* durante o dia. Já o horário noturno é mais favorável para a execução de testes remotos, pelo menor número de utilizadores na *cloud*.

Outra vantagem desta solução está relacionada com a possibilidade da execução de testes por qualquer interveniente (e não exclusivamente do *tester*), permitindo que múltiplos utilizadores realizem a execução dos testes. Deste modo os resultados são disponibilizados quando os intervenientes necessitam, e não necessariamente só quando o *tester* procede à execução dos testes.

1.4 METODOLOGIA DE TRABALHO

Para a concretização deste projeto foi adotada a metodologia de investigação orientada pela Engenharia (*The Engineering Design Process* [4]). A metodologia assenta numa série de passos seguidos pelos engenheiros de forma a encontrar uma solução para um problema. Os passos a serem seguidos são diversos, nomeadamente a definição do problema, investigação do tema, especificação de requisitos, *brainstorm* de soluções e escolha da melhor, desenvolvimento, protótipo e testes da solução, e por fim, se a solução não for de encontro com os requisitos estabelecidos, o processo é realizado novamente [4]. Deste modo é considerado que o método de engenharia é iterativo.

De acordo com a metodologia seguida, o problema em questão foi definido pela equipa. A recolha de dados foi realizada pela interpretação dos conteúdos anteriormente desenvolvidos, pela investigação e leitura de referências bibliográficas e informação credível disponível na internet. A par das etapas de especificação de requisitos, criação de soluções, desenvolvimento e protótipo de solução foram criadas tarefas para acompanhamento das mesmas. Estas tarefas foram calendarizas e registadas no *Team Explorer,* associado ao projeto do *Team Foundation Server* (TFS) [5], permitindo que o programador conseguisse obter *feedback* do número de tarefas que foram realizadas,

que estão em realização e que esperam ser realizadas. Estas tarefas, que podem distinguir-se em categorias, como *tasks* ou *bugs*, podem ainda ser associadas a cada integração de código do repositório de dados. Assim, o desenvolvimento é documentado a cada passo, prevendo que a pesquisa de detalhes de implementação possa ser realizada posteriormente com maior facilidade.

Após os testes e avaliação da solução criada, foram analisados os resultados para aferir se estavam de acordo com o pretendido, e se era vantajoso ou não uma nova iteração para melhoramento de componentes mais específicas.

1.5 CONTRIBUIÇÃO

Nesta dissertação é detalhadamente descrito como desenhar e implementar uma arquitetura que permite a execução de testes de *software* de forma automatizada, por múltiplos intervenientes do desenvolvimento, num ambiente distribuído de *Cloud Computing.* Esta solução segue uma abordagem de *Continuous Integration* (CI), e procura otimizar a distribuição de recursos, diminuindo a sobrecarga do *provider* de *Cloud Computing*, principalmente quando não houver testes em execução.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

A presente dissertação está dividida em sete capítulos, sendo o presente capítulo destinado ao enquadramento do problema e à contribuição da solução conseguida nesta dissertação. O segundo capítulo diz respeito ao SIJ, detalhando a estrutura do sistema e também da execução dinâmica de testes. O terceiro capítulo apresenta o estado de arte, descrevendo conceitos teóricos que apoiam a arquitetura proposta. No quarto capítulo são expostos todos os detalhes de análise e modelação da arquitetura e o no capítulo cinco são redigidos todos os detalhes de implementação. Nestes capítulos estão disponíveis vários diagramas para ajudar a entender todo o processo. No sexto capítulo são apresentados os resultados da aplicação da solução implementada, seguindo-se do capítulo sete que expõe as conclusões, problemas encontrados e trabalho futuro. Este documento termina com a bibliografia consultada

e os anexos que ilustram os diagramas expostos ao longo deste documento, porém com mais detalhe.

2 SISTEMA DE INFORMAÇÃO DA JUSTIÇA DE CABO VERDE

O presente capítulo apresenta o SIJ, do ponto de vista geral, e detalha partes do funcionamento interno e comunicação com sistemas externos, essenciais do SIJ. Este capítulo explica ainda a metodologia de desenvolvimento e dos testes de *software*, de modo a ser possível entender a arquitetura obtida durante a concretização desta dissertação.

2.1 VISÃO GERAL DO SIJ

O Sistema de Informação da Justiça de Cabo Verde é um sistema para a desmaterialização, total ou parcial, dos processos em tramitação nos Tribunais de Cabo Verde. Visualmente, é um sistema *web* de suporte aos utilizadores para ações de gestão e manuseamento de processos penais e civis de Cabo Verde [2] [6]. A interface *web* de apresentação correspondente à gestão dos processos penais e civis está ilustrada na Figura 1.

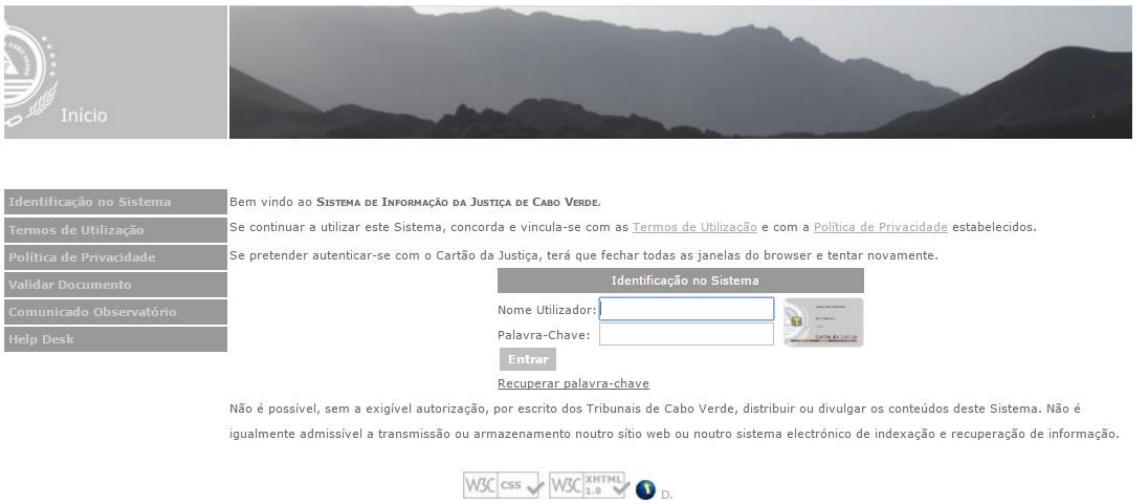

Figura 1 – Homepage do SIJ

O SIJ é atualmente composto pelas seguintes componentes: Sistema de Informação do Processo Penal (SIPP), Sistema de Informação do Processo Civil (SIPC), Diário de Justiça Eletrónico (DJE), Ordem dos Advogados de Cabo Verde (OACV), Portal Operacional de Tecnologias da Informação (iTop), Identificador Único de Justiça. Permite, ainda, a comunicação com a Direção Geral de Contribuições e Impostos (DGCI), Registo Nacional de Identificação (RNI), Sistema de Informação e Investigação Criminal (SIIC), entre outros. A Figura 2 foi desenhada pela equipa de desenvolvimento, sendo uma representação de todas as componentes do SIJ, algumas já completamente desenvolvidas, algumas em desenvolvimento e outras por desenvolver.

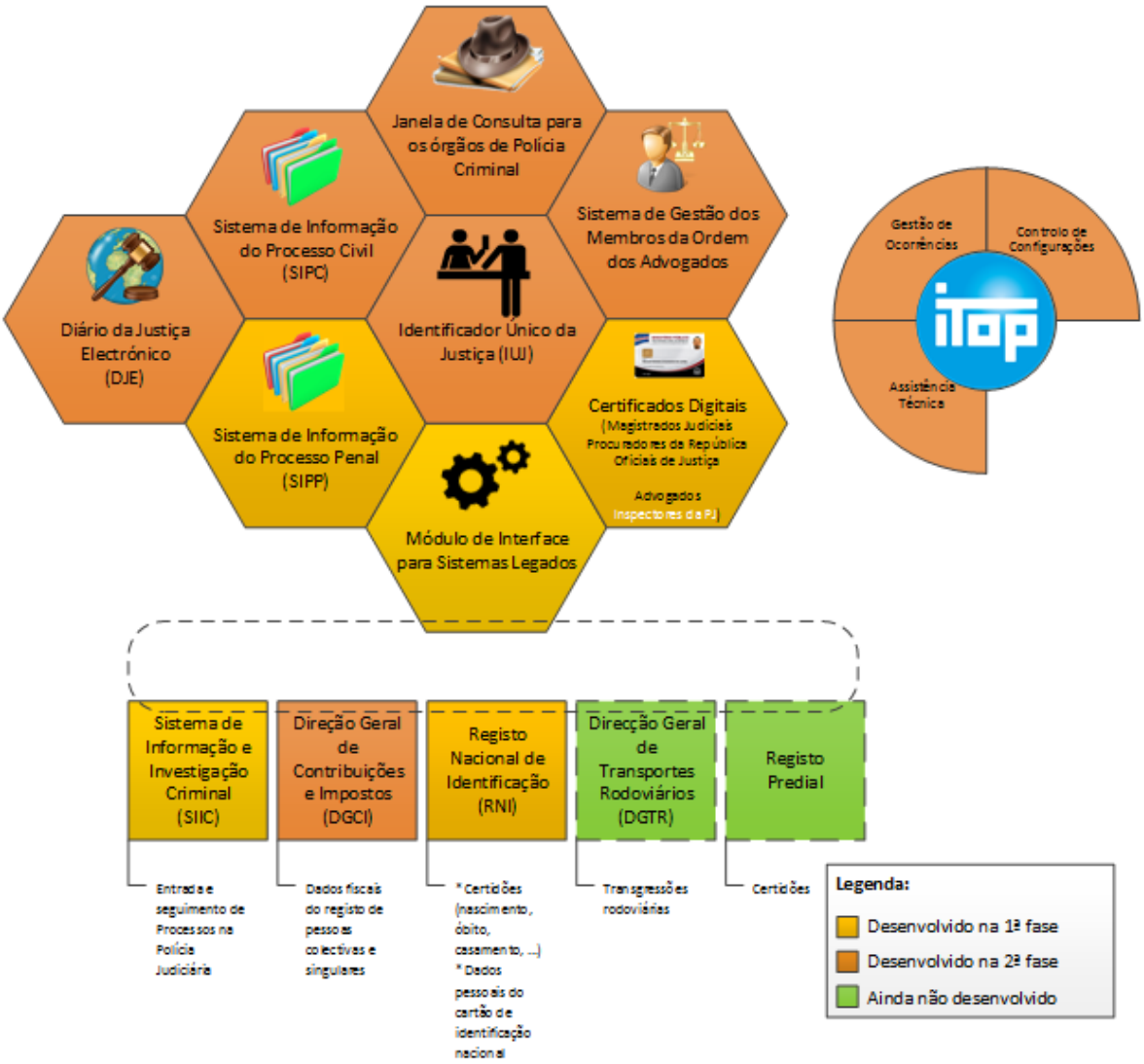

Figura 2 – Componentes SIJ

O SIPP e o SIPC partilham a mesma estrutura lógica, estando integrados na mesma aplicação *web* (representada pela Figura 1). Nesta dissertação é utilizada a componente SIPP como exemplo, visto ter sido a primeira componente desenvolvida no SIJ.

A arquitetura destas componentes (SIPP e SIPC) pode ser dividida em 4 camadas: camada de interface, camada de serviços, camada de lógica de negócio e a camada de base de dados (Figura 3). Esta figura foi inspirada numa ilustração apresentada no artigo "Assisted On-Job Training" [7], atualizando a componente da camada de serviços.

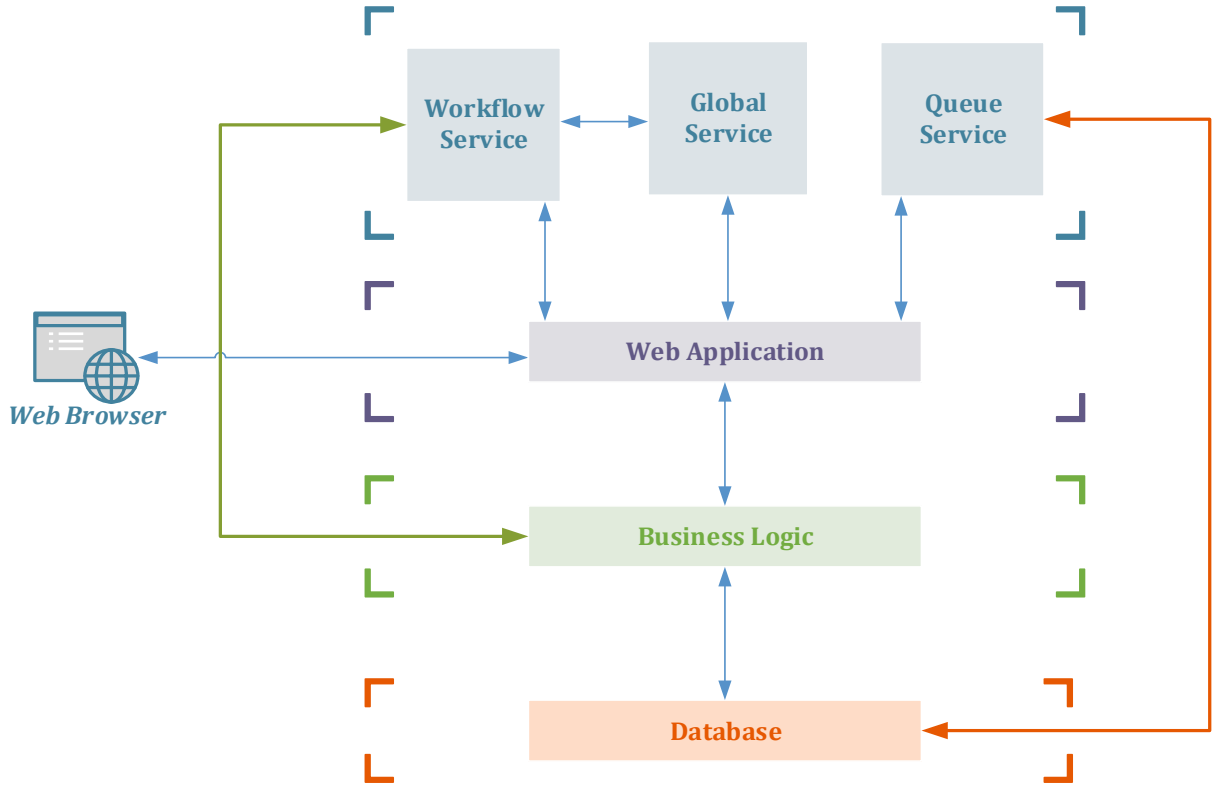

Figura 3 – Estrutura arquitetural

A camada de interface, ou aplicação *web*, permite a interação dos utilizadores com o sistema, onde são realizadas ações sobre processos civis e penais da justiça de Cabo Verde. Esta aplicação permite realizar um leque alargado de tarefas como a interação com processos judiciais, gestão de notificações e mensagens internas trocadas entre utilizadores, acesso a documentação legal, gestão de anotações de utilizadores, calendarização de tarefas, etc. [2].

A camada de negócio, a mais complexa, permite que a aplicação *web* e a camada de serviços partilhem o mesmo código base, isto é, nesta camada está todo o processamento de acesso à base de dados, lógica de validação, lógica de segurança, entre outros [2].

A camada de base de dados está implementada recorrendo ao sistema de gestão de base de dados *SQL Server* 2012 [8], sendo utilizado o serviço de mapeamento disponibilizado pela *Entity Framework* [9] como elemento de ligação entre a camada de base de dados e a camada da lógica de negócio.

Como é possível observar Figura 3, existe uma camada de serviços que faz a gestão de três serviços, serviço de workflow, serviço de base de dados e serviço global. Os serviços comunicam com a aplicação *web*, comunicação realizada segundo uma arquitetura orientada a serviços (SOA) [10]. O serviço de *workflow* gere todos os procedimentos envolvidos nos processos judiciais, mais concretamente, gere todos os estados, transações e interações que podem ocorrer em cada processo judicial (por exemplo, define o tipo de *roles* de utilizador num processo que se encontra em determinada fase, ou disponibiliza tipo de decisões possíveis por fase processual, etc.) [2]. Cada instância de um *workflow* representa um único processo judicial numa fase específica [2]. Os *workflows* são processos de longa duração que assentam sobre serviços do *Windows* (utilizando a *Framework Windows Communication Foundation* [11]). Estes serviços permitem a comunicação entre as diversas instâncias de *workflows* criadas ao longo do processo. O sistema de *workflows* disponibiliza uma interface de programação da aplicação (API) para interação com os *workflows*, que assentam na *Framework Windows Workflow Foundation.*

O serviço da base de dados permite gerir as tarefas que são executadas pela mesma. Anteriormente ocorriam muitos erros na base de dados devido a problemas de *deadlocks*, e para evitar o bloqueio sobre as tabelas da base de dados, foi desenvolvida uma estrutura, em forma de serviço *Windows*, que permite que tarefas pendentes esperem pela sua vez, numa fila de tarefas, de modo a serem executadas sequencialmente.

O terceiro serviço tem um papel de serviço global, permitindo gerir funcionalidades globais do sistema, como o serviço de mensagens curtas (SMS), persistência ou sincronização.

2.1.1 Metodologia de desenvolvimento

No decorrer do desenvolvimento do SIPP, os programadores foram divididos em três grupos: equipa de desenvolvimento da camada de apresentação (interfaces *web*), equipa de desenvolvimento das camadas base (camada de serviços, camada da lógica de negócio e camada de base de dados) e equipa de desenvolvimento de testes de *software* [2]. O desenvolvimento ocorre sobre uma metodologia iterativa, produzindo, idealmente, uma iteração de duas em duas semanas. O SIPP está desenvolvido sobre a *Framework .NET*, tendo como gestor do ciclo de vida do *software* o *Team Foundation Server* (TFS) [5], e este suporta todo o processo de desenvolvimento, produzindo *backlogs* categorizados em "*Users Stories*", "*Tasks*", "*Bugs*", "*Test Cases*" e "*Issues*" [2].

A metodologia de desenvolvimento deve conter, a cada nova interação, duas versões da aplicação ativas, uma versão para desenvolvimento, onde são adicionadas funcionalidades e corrigidos pequenos problemas, e uma versão de produção, que está sujeita a possíveis alterações de funcionalidades importantes, testes e novos *deploys* [2].

2.1.2 Integração com outros sistemas

Como descrito anteriormente, o SIJ integra com sistemas externos, consumindo e disponibilizando informações. No SIPP, há uma comunicação com a Rede Estatal de Cabo Verde, para partilha de dados do Registo Criminal de cabo-verdianos (comunicação bidirecional). É utilizado o serviço de correio eletrónico para comunicação com o utilizador (comunicação unidirecional). O SIPP comunica ainda com aplicações *web* mais recentes, como a OACV, numa comunicação bidirecional entre o SIPP e a mesma.

Como é possível ver na Figura 4, no SIPP, a entrada de dados pode ser realizada pelo sistema ou a partir de dados disponibilizados pela Polícia Nacional (PN) e Polícia

Judiciária (PJ). Esta comunicação é unidirecional. A comunicação entre o SIPP e o Registo Criminal é bidirecional, porque o SIPP recebe informações do registo criminal de um arguido e pode ter a necessidade de atualizar este mesmo registo de acordo com a decisão sobre o arguido.

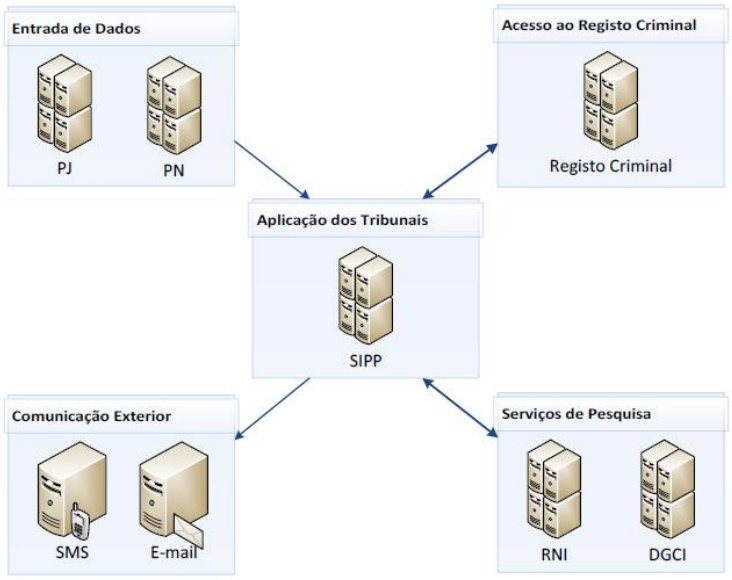

Figura 4 – Interação entre os diversos sistemas [6]

O SIPP utiliza serviços de correio eletrónico e mensagens para notificar os utilizadores do sistema de alguma alteração relevante, e esta comunicação é unidirecional. Por fim, o SIPP utiliza serviços de pesquisa de cidadãos pois não possui efetivamente dados dos mesmos. Assim sendo, utiliza o RNI e guarda uma referência da entidade no SIPP.

Em 2014 foram desenvolvidos, pela equipa de desenvolvimento do SIJ, dois novos sistemas que interagem com o mesmo, a OACV e o DJE.

2.1.2.1 Ordem dos Advogado (OACV)

O SIPP interage com a OACV para realizar pedidos de advogados e associar os mesmos a um processo penal. Quando o SIPP faz o pedido, envia o número do processo, a comarca e os sujeitos processuais (arguido, ofendido, testemunha) associados ao processo. Posteriormente o bastonário delega a função a um advogado oficioso para integrar no processo do SIPP.

A OACV permite interação com três tipos de utilizadores: o administrador, o advogado e o bastonário. O utilizador "administrador" tem opção de gestão de utilizadores (gestão de dados pessoais, domicílios profissionais e quotas), de turnos, de lista de oficiosas, autorizações e processos.

O utilizador "advogado" pode gerir os dados pessoais, domicílios profissionais, comarcas por turnos, comarcas para oficiosas, processos com oficiosas, quotas e situações. Na Figura 5 está ilustrada a página correspondente à gestão de dados pessoais de um advogado.

Figura 5 – Página do utilizador "advogado" da OACV

Já o utilizador "bastonário" pode criar processos na OACV, atribuir sujeitos processuais a um processo, e nomear os advogados para os sujeitos processuais.

2.1.2.2 Diário de Justiça Eletrónico (DJE)

O DJE é considerado um sistema que permite aproximar os cidadãos à justiça de Cabo Verde e permite uma maior transparência entre todas as entidades envolvidas. Permite ver a agenda de eventos nacionais, os editais, validar documentos, a listagem de turnos por comarca, o movimento processual, a tramitação processual, a distribuição de processos, a listagem de advogados oficiosos por comarca, a doutrina e jurisprudência, os relatórios, estatísticas e concursos. A página inicial do DJE está ilustrada na Figura 6.

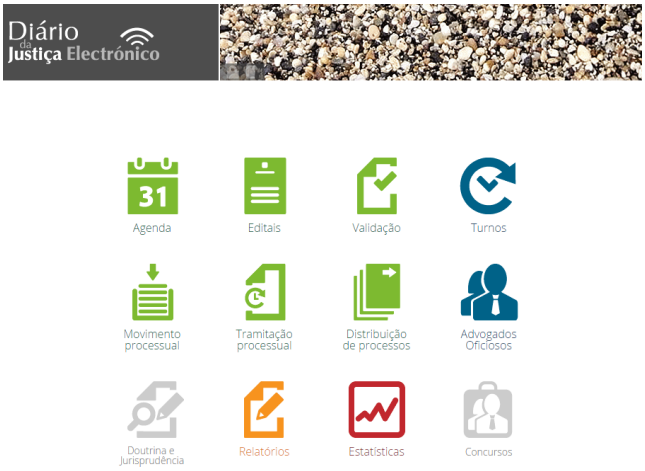

Figura 6 – Homepage do Diário de Justiça Eletrónico

2.2 METODOLOGIA DE TESTES NO SIPP

Neste subcapítulo será abordada a estratégia atual de execução de testes na versão de desenvolvimento do SIPP. Inicialmente irá ser explicado o conceito e tipos de testes de *software* e posteriormente explicado o processo atual da execução dinâmica de testes no projeto do SIPP.

2.2.1 Testes de *software*

Os testes de *software* são essenciais para conhecer o estado funcional de um sistema de informação. Na terceira edição do livro "*The Art of Software Testing*" [3] é mencionado que "testar o *software*" é cada vez mais difícil mas ao mesmo tempo é cada vez mais fácil. Mais difícil porque ao longo de várias décadas a tecnologia evoluiu, criando novas linguagens de programação, utilizadas em diferentes sistemas operativos, sobre diferentes peças de *hardware*. Isto tem como consequência um aumento da responsabilidade sobre quem produz sistemas ou aplicações, pois estes chegam cada vez mais rápido e a mais pessoas, podendo influenciar a vida das mesmas. Mas também se torna cada vez mais fácil porque estas novas tecnologias e estruturas estão mais capazes para se adaptarem umas às outras, com muito mais facilidade, fazendo uso de bibliotecas ou outras soluções já existentes, sem a necessidade do próprio programador desenvolver tudo de raiz.

Um teste de *software* é um processo, ou uma série de processos, desenhado para garantir que o código faz o que foi projetado para fazer. Ao mesmo tempo que garante o correto funcionamento, o teste não deve fazer nada de forma não intencional. Os autores do livro defendem que não é possível testar todas as combinações possíveis de um programa, porque, por exemplo, numa aplicação complexa, a concretização destas combinações demoraria muito tempo a ficar concluída. Nesta perspetiva, o *tester* necessita de ter a atitude certa para testar uma aplicação, atitude que deve estar focada em realizar os testes para encontrar erros e não com a atitude que todos os testes necessitam de passar. Mais concretamente, o *tester* deve partir do prossuposto que o programa contém erros e a sua tarefa é explorar as possibilidades dos mesmos ocorrerem, para ser então possível a sua correção, acrescentando valor e qualidade ao programa [3].

Jiantao Pan [12] classifica os testes de *software* em três categorias: objetivos, ciclo de vida do *software* e âmbito de teste. A classificação por objetivos separa a classificação em quatro tipos: testes de exatidão, *performance*, fiabilidade e segurança. A classificação pelo ciclo de vida do *software* divide os testes por fase de requisitos, testes manuais e automatizados e testes de aceitação. E a classificação pelo âmbito de testes divide os testes em testes unitários, de integração, de componentes e de sistemas.

De um modo resumido, estes testes permitem verificar o correto funcionamento de componentes (métodos ou funções), conhecendo ou não o funcionamento interno das mesmas, validando os resultados obtidos nos testes. Os testes permitem verificar a correta integração destas componentes simples até à integração do sistema como um todo. Permitem também validar as componentes em qualquer fase de desenvolvimento, desde a modelação até à fase de interação entre o utilizador e a aplicação. Por fim, também permitem a simulação de falhas sobre o sistema validando a sua segurança e fiabilidade [2] [12]. Na Figura 7 está representado um esquema resumido da classificação de testes abordada anteriormente [2].

Classificação de testes	Classificação por objetivos	Testes de exatidão
		Testes de performance
		Testes de fiabilidade
		Testes de segurança
	Classificação pelo ciclo de vida do *software*	Testes na fase de requisitos
		Testes manuais e automatizados
		Testes de aceitação
	Classificação pelo âmbito de testes	Testes unitários
		Testes de integração
		Testes de componentes
		Testes de sistema

Figura 7 – Classificação de testes

2.2.2 Metodologia de execução de testes no SIPP

Atualmente, a versão de desenvolvimento do SIPP conta com uma execução distribuída dos testes. A equipa de testes trabalha sobre uma arquitetura que permite executar os testes de forma distribuída, num ambiente de *Cloud Computing*, em VMs, apoiando-se num *deploy* automatizado para a configuração dos testes nas mesmas máquinas. O SIPP está desenvolvido sobre a *Framework .NET*, tendo uma configuração de testes dinâmica, utilizando um controlador e agentes de testes do *Visual Studio* (VS). O TFS possui um sistema de controlo de versões (SCV) integrado que permite fazer toda a gestão da versão de ficheiros utilizados pela equipa de desenvolvimento.

Cada VM tem o papel de "agente de testes", contendo a configuração necessária para a execução dos testes e está sobre a *Framework* OpenNebula. Atualmente as VMs têm um número fixo, sendo sempre as mesmas, e o ambiente delas foi configurado manualmente pela equipa de testes, no momento da sua criação. Estas VMs estão sempre ligadas na *cloud*, consumindo os recursos da mesma, mesmo quando os testes não estão em execução. A configuração manual do ambiente das VMs contempla a configuração de ficheiros do servidor *web*, do *SQL Server* (serviços, instância *SQL* e utilizadores), instalação dos agentes de testes do VS, e criação de serviços que o SIJ utiliza para todo o funcionamento do sistema.

Os testes de software no SIPP estão divididos por testes à camada lógica de negócio e à camada de base de dados, testes aos *workflows* e testes de interface. Os testes unitários são realizados em métodos ou funções e também em *triggers* e *stored procedures*. Os testes aos *workflows* são mais complexos porque envolvem a criação de uma instância de *workflow*, para ser possível alterar o *workflow* para a fase pretendida. Assim, estes testes garantem que, para cada *workflow*, todas as transações entre estados funcionam conforme o esperado e que todos os eventos (em cada estado) desempenham a função que lhes foi delegada [2].

Os testes de interface *web* verificam o comportamento da camada de interface, simulando a interação entre o utilizador e a interface *web*. Isto significa que a simulação tem de ser bem preparada, gerando previamente a informação necessária ao teste, para ser possível validar a componente no momento desejado. Deste modo é necessário que seja estabelecida a comunicação com todas as camadas da aplicação, para que seja possível ao *tester* iniciar o teste exatamente no estado que deseja. Os testes de interface são designados como *Coded UI Test* [13] e as ações têm de ser realizadas no *browser Internet Explorer.* Estas ações são gravadas pelo *software*, gerando o código para um ficheiro na solução do projeto.

2.2.2.1 Execução distribuída de testes no SIPP

No primeiro capítulo desta dissertação foi mencionado que os testes passaram a executar de 10 a 14 horas para 2.5 a 3 horas devido à execução dos mesmos acontecer num ambiente distribuído. A solução distribuída conta com uma VM que simula o papel de domínio de rede, denominada de "**opennebula.ua.pt**", onde estão registadas todas as VMs que participam no processo de execução de testes. O servidor tem o sistema operativo *Windows Server 2008 R2 Enterprise*, e tem instaladas todas as dependências necessárias para executar a solução do SIPP (*framework .NET*, com o VS 2013 e o *SQL Server* 2012). As VMs que atuam como agentes e controlador de testes estão no mesmo domínio e permitem que os testes sejam executados remotamente, de forma automatizada e paralela. O controlador de testes liga-se diretamente ao VS instalado no servidor, dando a possibilidade ao *tester* de confirmar visualmente e de forma simples que os agentes se encontram ligados e disponíveis para execução dos testes,

como visível na Figura 8. É possível observar que o controlador tem associado a si catorze agentes de testes. Neste cenário as VMs estão desconectadas, e o *tester* tem a possibilidade de reiniciar todas as VMs ou apenas as que desejar.

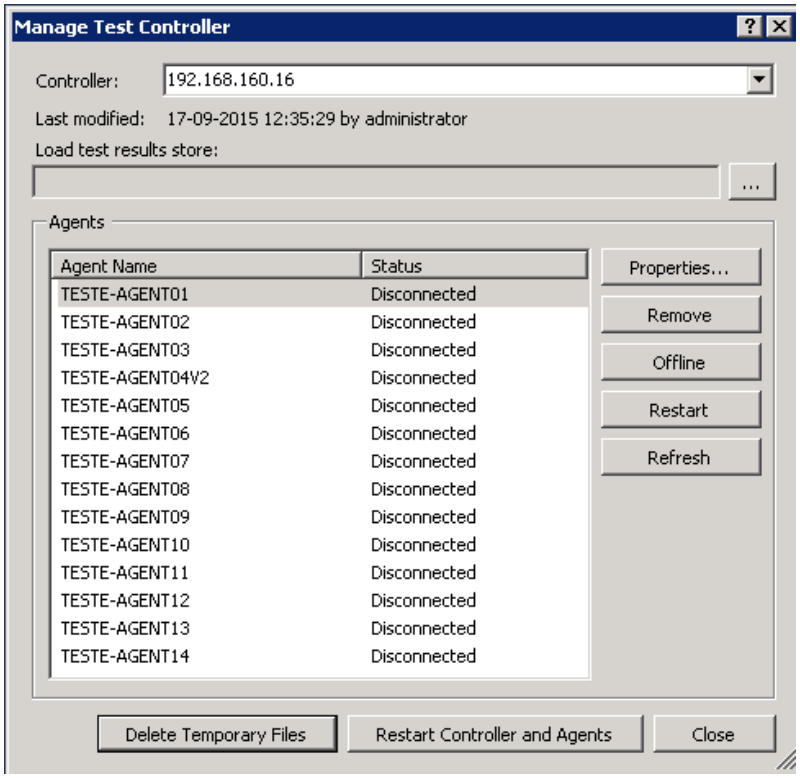

Figura 8 – Gestor de controlador de testes

Os agentes de testes recebem os pedidos de execução de testes, executam os mesmos e recolhem os resultados enviando-os para o controlador [2]. A esquematização é representada na Figura 9.

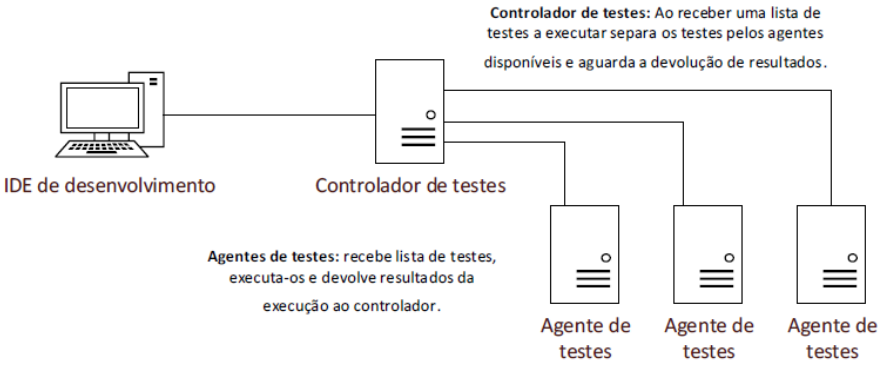

Figura 9 – Controlador e agentes de testes [2]

A solução do SIPP tem vários projetos de testes distintos e cada um deles contém um ficheiro de configuração. De modo a evitar que os agentes de testes utilizem todos a mesma base de dados (por exemplo, a instância de base de dados instalada no servidor de domínio, e todos acediam a essa mesma instância), foi definido que em cada um estaria uma configuração isolada. Assim, cada VM tem uma configuração autossuficiente, contendo todos os elementos necessários para a execução dos testes.

O *deploy* local, por agente de testes, de todos os elementos necessários ao funcionamento do SIPP (servidor *web*, serviços globais, de base de dados e de *workflows* e instância de base de dados), possibilita que cada agente de testes possa executar todos os testes utilizando apenas recursos locais (base de dados local, *localhost*). Isto significa que o agente de testes necessita de ter: a aplicação *web* alojada na instância de serviços de informação da internet (IIS) local, os serviços de gestão de *workflows,* funcionalidades globais e base de dados instalados como serviços do *Windows* e uma instância do *SQL Server* com a versão adequada da base de dados da aplicação. Assim, todos os ficheiros de configuração de todos os projetos de testes necessitam de conter as definições de acesso aos recursos locais de cada agente de testes (*localhost* para acesso a aplicação *web*, ".\SQLEXPRESS2K8FTS" para acesso a base de dados) [2].

As VMs de agentes e controlador de testes são recursos virtualizados no OpenNebula, e encontram-se registados no domínio de rede referido anteriormente, facilitando bastante o processo de *deploy* do SIPP pelos agentes de testes. O servidor controla as permissões e segurança de todos os computadores que estão ligados ao domínio, e estes computadores estão registados num controlador de domínio (*Active Directory (AD)*). Neste controlador são guardadas as informações dos utilizadores, computadores, e outros recursos da sua rede, ajudando a ter uma administração das informações de forma segura e centralizada [14]. Deste modo são evitados problemas relacionados com acesso remoto de serviços e pastas partilhadas, servidores *web* (IIS) e instâncias de base de dados [2].

O computador do domínio da rede também é utilizado pela equipa de testes para experiências, como por exemplo testar a comunicação entre a própria máquina e um

agente de testes. Estas comunicações podem ser testadas pela invocação de um *script Windows Powershell*[1] que permita copiar os ficheiros de uma máquina para a outra.

2.3 CONSIDERAÇÕES FINAIS

A estrutura atual do sistema SIJ está bem consolidada e está em constante desenvolvimento, estando neste momento a ser desenvolvida a componente do Processo Civil.

As equipas de desenvolvimento e testes trabalham todos os dias para colmatar problemas que são encontrados pela equipa de *helpdesk* de Cabo Verde, que avança com a gestão dos processos penais no sistema.

A estrutura da execução dos testes de *software* do SIPP é uma estrutura bem conseguida e pode ser melhorada se a componente de gestão das VMs for automatizada, no que diz respeito à rentabilização da *cloud* e elasticidade dos recursos da mesma. Esta rentabilização dos recursos tem como intenção distribuir os recursos consumidos pelos agentes de testes quando os testes não estão em execução, de forma otimizada, para suporte a outros utilizadores, como alunos da Universidade de Aveiro ou equipas de investigação.

[1] A Windows *Powershell* é uma consola muito utilizada pela equipa de testes para comunicação com as VMs remotas. Por definição, a *Powershell* é uma consola por linha de comandos, que executa comandos, funções, filtros, scripts (.ps1) e aplicações (.exe) [43].

3 ESTADO DE ARTE

3.1 CONTINUOUS INTEGRATION (CI)

De acordo com o conteúdo apresentado em "*Continuous Integration using Cloud Computing*" [1], *Continuous Integration* (Martin Fowler, 2000) é uma prática de desenvolvimento de *software* onde membros de uma equipa integram o seu trabalho frequentemente durante o dia, pelo menos uma vez. Cada uma destas integrações é verificada por uma compilação automática e pode ser validada por testes de integração.

O processo de CI é apresentado na Figura 10.

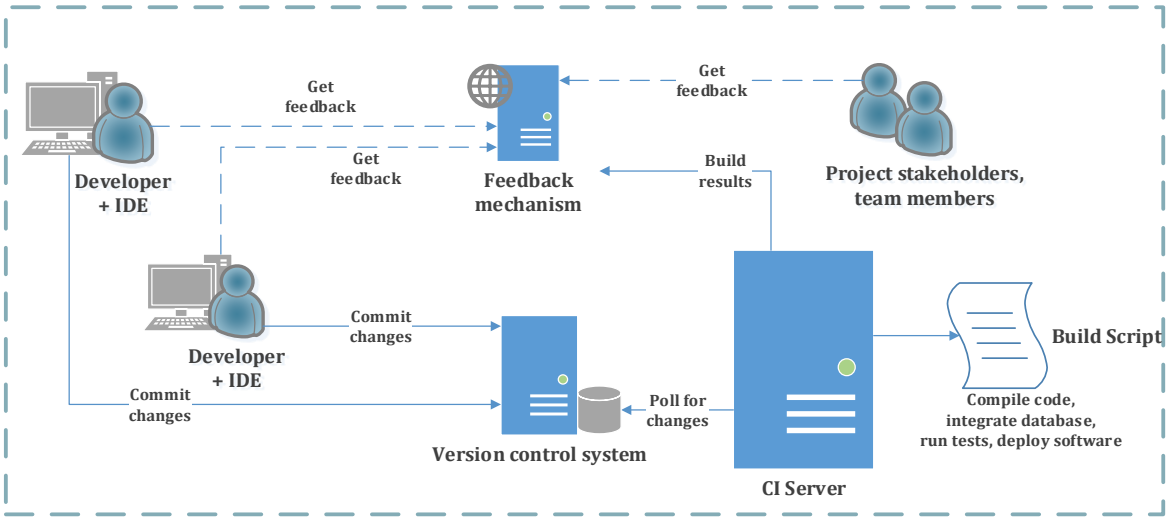

Figura 10 – Processo de CI

Na figura anterior é possível verificar que existem atores chave que interagem entre si, que são os programadores, sistema de controlo de versões, servidor de CI, *script* de compilação e sistema de *feedback*, e estes atores permitem uma execução válida do processo de CI. O programador, depois de concluir uma tarefa, compila a solução para verificar que não ocorreram erros, e depois integra as mudanças no repositório. O SCV permite gerir o código e outros aspetos relacionados com o desenvolvimento de *software,* e a execução de CI é sobre a versão principal do repositório. O servidor de CI interage com o SCV, pesquisando por mudanças consideradas significativas, e se elas

existirem, executa uma compilação e testa o código. O *script* de compilação é usado para compilar, testar, inspecionar ou publicar o *software*. Por fim, o mecanismo de *feedback* torna possível ver, de imediato, o resultado obtido.

A integração de *software* pode não ser um problema num projeto com dimensões pequenas e com poucas dependências externas, mas à medida que a complexidade de um projeto cresce, é necessário integrar e assegurar que as componentes de um *software* funcionam juntas de forma coesa. A opção de esperar pelo fim do desenvolvimento pode levar à fraca qualidade do *software* e a resolução de potenciais problemas terá um custo mais elevado, e consequentemente poderá atrasar a entrega do produto aos clientes. A adoção desta prática permite que a equipa de desenvolvimento sinta uma maior confiança no produto e mais segurança em cada integração de código. Pretende-se que o conceito de compilação seja visto numa perspetiva mais ampla, isto é, não referente apenas ao processo de compilar o código, mas sim todo um processo de integração, compilação, teste, e *deployment* do *software* [15].

Existem algumas estratégias de validação no mecanismo de CI para a compilação do projeto, nomeadamente contínua, diária, semanalmente, entre outras [16]. Deste modo podem reduzir-se significativamente problemas de integração encontrados frequentemente pela operação de *merge* [17]. Com a realização das operações de *merge, update* e *check-in*/*commit* todos os dias, as integrações poderão ser mais pequenas e menos complexas, e será mais fácil de corrigir os conflitos. Uma das estratégias possíveis em CI é, por exemplo, garantir que o repositório é compilado antes e depois de cada *check-in*. Assim, caso após o *check-in* de código, o projeto em questão, já não compilar, o programador deve tomar a correção do erro como prioridade. Tipicamente, além da simples compilação de código, podem também ser executados testes unitários para despistagem de eventuais problemas.

Martin Fowler enumera as **boas práticas** que devem ser seguidas pela equipa para a prática de CI:

◊ *Manter um único repositório de código*: Num projeto de *software*, os ficheiros devem ter operações de *update*/*revert* e possibilidade de ver o histórico das versões dos mesmos. Estas versões são geridas por ferramentas que são parte integral do desenvolvimento. As ferramentas, que não têm todas as mesmas opções ou o mesmo nome, têm como principal objetivo a gestão de versões de código;

◊ *Automatizar a compilação*: Algumas ferramentas de desenvolvimento de *software* integram compiladores automatizados, como é o caso da *framework .NET*, que integra no seu ambiente de desenvolvimento a ferramenta *MSBuild*. Com ferramentas como esta é possível compilar e lançar o sistema com um único comando. O compilador tem um *script* associado que deve conter todos os comandos necessários para compilação e execução do sistema em qualquer máquina;

◊ *Fazer a compilação self-testing*: Um programa pode executar mas isso não garante que seja de forma correta. Para prevenir que ocorram erros deve ser feita a inclusão de testes automáticos no processo de compilação;

◊ *Todos os programadores fazem commit para a versão main todos os dias*: O ciclo normal de *commit* passa por, previamente, fazer um *update* ao código, compilar, e se não ocorrerem erros (ou conflitos), o programador pode enviar as mudanças para o repositório. Se isto acontecer frequentemente durante o dia, os conflitos entre ficheiros modificados por programadores diferentes têm tendência a diminuir;

◊ *Cada commit deve compilar a versão main na máquina de integração*: Alguns fatores podem influenciar o mau funcionamento da compilação, como por exemplo, os programadores não fazerem o processo normal de *commit* (com *update* e compilação) ou então, por exemplo, o ambiente de desenvolvimento dos programadores ser diferente. Assim, deveria ser confirmado que a compilação realizada antes da integração funciona como o suposto, para então ser possível a atualização da versão *main*;

◊ *Resolver compilações falhadas imediatamente*: A CI surge para trabalhar numa versão mais estável de uma solução, e como tal, manter a solução sem erros é prioridade, e por isso, se há um erro numa compilação, tem de ser descoberto e resolvido o mais rápido possível;

◊ *Manter a compilação rápida*: O *feedback* deve ser disponibilizado rapidamente, isto é, se a compilação demorar muito tempo, devem arranjar-se mecanismos para acelerar o processo e disponibilizar o *feedback* aos programadores;

◊ *Testar o projeto num ambiente que simule do ambiente de produção*: É importante desenvolver e testar o projeto num ambiente clone ao ambiente produção, porque muitas vezes estes ambientes são diferentes. Deste modo, o programador pode evitar erros de integração relativamente ao ambiente de produção;

◊ *Tornar mais fácil para qualquer um obter a última versão executável*: Todas as pessoas envolvidas no sistema devem conseguir ver a última versão de produção do sistema, porque a mesma pode ser necessária para fazer desmonstrações ou testes exploratórios;

◊ *Todos podem ver o que está a acontecer*: A prática de CI pretende fortalecer as integrações do sistema e a comunicação entre as pessoas envolvidas. Assim, é essencial que qualquer membro tenha conhecimento do estado do sistema e das mudanças que são feitas ao longo do tempo;

◊ *Automatizar o deployment do projeto*: Às vezes, para a prática de CI, pode ser necessário ter múltiplos ambientes de testes (como o exemplo das máquinas virtuais de testes) e como tal o *deployment* automático nas mesmas é algo muito útil, pela facilidade que proporciona. Assim, o sistema deve ter um *script* que permita facilmente fazer o *deploy* da aplicação em qualquer ambiente de desenvolvimento.

Martin considera que o melhor benefício que a prática CI dá é a redução do risco. Isto porque com sua utilização, os programadores já não investem tanto tempo na

integração, já sabem em que ponto está o sistema, e sabem, no momento, se as suas contribuições para o sistema causam ou não erros, e se sim, então corrigem-nos antes de causar maiores erros no sistema completo. No livro *Continuous Integration: Improving Software Quality and Reducing Risk* [15] é explorado mais pormenorizadamente o valor de CI. Da mesma opinião de Martin, os autores do livro afirmam que o valor de CI é a redução de risco, a redução de processos manuais realizados repetidamente, a produção de *software* estável pronto em qualquer momento, uma melhor visibilidade do projeto e o estabelecimento de maior confiança no produto. Ao fazer várias integrações diárias reduzem-se riscos como a descoberta tardia de erros, a falta de *software* coeso, de baixa qualidade e visibilidade do projeto. CI gera *software* pronto a publicar em qualquer momento, sendo este um dos aspetos mais relevantes da adoção desta prática. Por fim, no livro é mencionado que CI estabelece maior confiança no produto, porque a cada compilação, a equipa de desenvolvimento conhece o impacto das mudanças realizadas no código, e obtém os resultados dos testes executados, verificando o comportamento do *software*, para a obtenção de um produto funcional e testado. Assim, CI permite a vigilância do estado do produto, várias vezes por dia, reduzindo o tempo de aparecimento de um defeito e o tempo em que o mesmo é corrigido, melhorando a qualidade de todo o *software* [15].

De modo a conseguir adotar esta prática para a solução pretendida, é necessário conhecer ferramentas disponíveis para ser possível escolher a que melhor se adequa ao problema.

3.2 FERRAMENTAS PARA CI

Para a concretização de uma arquitetura de CI é essencial a análise e escolha de um SCV, um servidor CI, um gestor de compilação e de testes. Pode ainda ser completado por um mecanismo de *feedback* e uma ferramenta de análise de código. Tendo em consideração o caso de estudo concreto, deve ser também avaliado se as ferramentas atualmente em utilização são as mais indicadas para a inclusão do processo de CI.

3.2.1 Sistema de controlo de versões (SCV)

Um SCV, ou também chamado de sistema de controlo de revisão, sistema de controlo de código, ou repositório de controlo de versões, permite ver o histórico do desenvolvimento de tudo o que foi uma vez adicionado ao repositório; permite criar/editar/eliminar ficheiros, fazendo *update* ou *reverse*, em qualquer momento no tempo e de qualquer ficheiro no repositório, permitindo a comunicação entre os membros da equipa; permite comparar as versões de ficheiros locais com ficheiros que estão no servidor, pesquisando por data, ou *label*/código associado ao *commit* no repositório. Não guarda apenas ficheiros de código, contendo também ficheiros binários, esquemas de base de dados, *scripts* e ficheiros de configuração de serviços e testes. Este sistema é uma peça fundamental para a arquitetura de CI mas não só, isto é, é uma ferramenta essencial para o trabalho entre uma equipa independentemente da metodologia seguida.

Para a escolha de um SCV devem ser consideradas algumas características importantes sobres os sistemas: centralizado *vs.* distribuído; transacional *vs.* não-transacional; bloqueio *vs.* não-bloqueio de ficheiros; e grátis *vs.* licença [16]. De acordo com o livro *Continuous Integration in .NET* [16], a primeira característica que deve ser analisada é o fator custo-benefício, relacionado com as funcionalidades disponibilizadas nos sistemas grátis *vs.* com licença. Um outro aspeto importante é se o sistema é centralizado ou distribuído. Num sistema distribuído, o programador tem a sua própria versão de código e os *commits* que realiza são locais, sendo considerado um repositório privado. Posteriormente há uma versão pública do repositório, gerida por um administrador, que se encarrega de juntar todas as versões. Já um sistema centralizado é o oposto e todas as pessoas trabalham sobre a mesma versão, no mesmo servidor. Normalmente estes sistemas são mais adequados para uma arquitetura de CI, por serem mais fáceis de incorporar e requererem menos conhecimento sobre a infraestrutura do repositório. Relativamente ao tópico de um sistema ser ou não transacional, já não é tanto um problema dos sistemas modernos mas é tão importante como os outros aspetos, e diz respeitos às operações serem atómicas, como por exemplo, um *commit* ser feito como um todo. Isto significa que se existir algum problema durante a operação, o sistema ficará no estado anterior à tentativa de

commit, e este não é realizado. Por fim, um sistema que permita o bloqueio de ficheiros permite que uma pessoa fique com um ficheiro para si, impedindo outro membro de editar esse mesmo ficheiro. Já o não bloqueio de ficheiros permite que seja possível editar os mesmos em todas as ocasiões, podendo levar a conflitos.

Foram analisados três sistemas que permitem fazer a gestão de controlo de versões.

3.2.1.1 Team Foundation Server (TFS)

O TFS fornece um núcleo de funcionalidades para as equipas de desenvolvimento, como gestão de projeto, acompanhamento de itens de trabalho, gestão de casos de testes, automação de compilação, relatórios, gestão de laboratório, ambiente de gestão de *feedback* e controlo de versão, bloqueando a edição de um ficheiro para um único utilizador [18]. É uma ferramenta muito completa, que disponibiliza funcionalidades para integração com o VS, contendo assim uma janela de controlo de versões, e explorador da equipa, que permite ver as funcionalidades descritas acima. É uma ferramenta desenhada para qualquer interveniente do desenvolvimento, como programador, *tester*, arquiteto e gestor do projeto. O TFS é um sistema que necessita de chave de ativação/licença.

3.2.1.2 Subversion + TortoiseSVN

O *Subversion* [19] é um sistema de controlo centralizado (à semelhança do TFS). É um sistema que está estável e é utilizado em muitas organizações, é *open source* e grátis. Os programadores trabalham a partir do seu ambiente e necessitam de acesso ao servidor apenas na altura de fazer *commit* para o servidor, permitindo que mais do que uma pessoa trabalhe sobre o mesmo ficheiro. Esta ferramenta consegue detetar a informação de versões de ficheiros, diretórios e metadata, isto é, consegue saber o histórico entre mudanças e renomeação de diretórios, ao contrário de outros sistemas. Tem um *commit* atómico, ou seja, ou todo o conteúdo é submetido ou então as mudanças não são enviadas para o servidor [19]. O *TortoiseSVN* é um cliente do *Subversion* para *Windows* que integra diretamente no *Windows Explorer* e permite as normais opções de um repositório.

3.2.1.3 Git + TortoiseGit

A ferramenta *Git* [20] é distribuída, ou seja, pode fazer-se *commit* para a versão local, sem haver a necessidade de ligação ao servidor. O projeto tem o seu próprio repositório (privado) e um repositório comum para os elementos da equipa (público). O *Git* permite que todos os programadores façam as operações habituais ou que haja um administrador que o faça. Os *commits* são reconhecidos com um identificador em vez de números ou *labels* de revisões. Em *Windows*, os utilizadores podem usar o *TortoiseGit*, que faz o controlo de versão do *Git*, implementado como uma interface de comandos *Windows* [21]. À semelhança do *Subversion*, é uma ferramenta grátis.

3.2.1.4 Resumo

Depois de analisadas as ferramentas, é necessário escolher uma delas. Os SCVs apresentados não se diferenciam muito entre eles no que diz respeito às funcionalidades principais que a generalidade destas ferramentas oferece. A Tabela 1 apresenta o resumo das características anteriormente especificadas.

	TFS	SVN + TortoiseSVN	Git + TortoiseGit
Sistema centralizado	**X**	**X**	
Gestor de ciclo de vida	**X**		
Integração com Visual Studio	**X**		**X**
Bloqueio de ficheiro	**X**		
Transacional	**X**	**X**	**X**
Open Source		**X**	**X**
Solução adotada	☑		

Tabela 1 – Resumo sistema de controlo de versões

O TFS é a ferramenta utilizada neste momento no desenvolvimento do projeto, é fácil de interagir e integra diretamente com o VS 2013 (IDE de desenvolvimento atual) e neste caso a equipa já está familiarizada com o seu funcionamento. Nesta perspetiva, o TFS foi escolhido como SCV para arquitetura em desenvolvimento.

3.2.2 Servidor de *Continuous Integration* (CI)

Um servidor de CI guia o processo de CI, verificando mudanças no repositório e coordenando os passos do processo. Idealmente é um servidor que executa algo predefinido a cada mudança do código e que entrega *feedback* imediato à equipa de desenvolvimento (tipicamente tem o seu próprio mecanismo de *feedback*). Normalmente estes servidores são conduzidos por um ficheiro de configuração (*script*) que especifica os passos a executar durante o processo de compilação e integração. É aconselhável ter uma máquina que apenas corra o servidor CI porque um processo corretamente criado deve ter o menor número de dependências possíveis [16]. Pode ser calendarizada uma compilação para ser, por exemplo, de hora em hora, ou uma vez por dia à mesma hora, entre outras opções.

Foram analisados três sistemas que se comportam com servidor de CI.

3.2.2.1 Team Foundation Server (TFS)

O TFS não é apenas um sistema de gestão de ciclo de vida que controla o repositório de código, é também um servidor de CI da *Microsoft* e como tal enquadra-se bem com tecnologias *.NET*. Permite orquestrar todas as componentes necessárias para o processo de desenvolvimento, desde o desenvolvimento do *software*, gestão de requisitos e projeto, desenvolvimento de testes e garantia de qualidade [22]. A compilação do servidor pode ser realizada pelo *MSBuild*.

3.2.2.2 CruiseControl.NET (CC.Net)

O CC.Net é um servidor de CI, implementado usando *.NET*, que pode executar com consola ou serviço *Windows*. Esta versão é baseada numa versão da ferramenta em *Java*. O servidor automatiza o processo de integração por monitorizar diretamente o repositório de código, validando as mudanças de atualizações do código [23]. O CC.Net pode gerir remotamente o processo usando um sistema chamado *CCTray*, que consegue aceder ao servidor para inicializar compilações ou apenas ser notificado da ocorrência de uma compilação. A compilação do servidor pode ser realizada pelo *MSBuild* ou *NAnt*.

3.2.2.3 Jenkins/Hudson

Jenkins é um servidor de CI baseado em *Java*, usado para projetos desde *.NET*, *Ruby*, *PHP*, *Grails*, e inclusive *Java* [24]. É uma tecnologia *open source*, tem diversos *plugins* para suporte à comunicação, execução de testes e integração com outros sistemas, e tem uma configuração simples baseada em interface *web*, tendo também a possibilidade de utilização da linha de comandos para executar as ações pretendidas. O *Jenkins* executa a compilação do projeto via uma funcionalidade chamada *job*, que consiste na execução de uma série de tarefas [25]. A compilação do servidor pode ser realizada pelo *MSBuild* ou *NAnt*.

3.2.2.4 Resumo

Os servidores apresentados registam algumas diferenças entre eles no que diz respeito à sua estrutura. No entanto, estas diferenças são mínimas se se considerar as funcionalidades que se pretendem usar. A Tabela 2 apresenta o resumo das características anteriormente especificadas.

	TFS	CC.Net	Jenkins
Gestor de ciclo de vida	**X**		
Integração com Visual Studio	**X**		
Documentação	**X**	**X**	**X**
Baseado em JAVA		**X**	**X**
Open source		**X**	**X**
Solução adotada	☑		

Tabela 2 – Resumo Servidor CI

Assim, dada a decisão anterior no caso de estudo, relativamente ao TFS, optou-se por mantê-lo como servidor de CI.

3.2.3 Gestão de compilação

Para realizar a compilação do código é possível escolher, entre outras opções, o *MSBuild* e *NAnt*. Permitem a utilização de *scripts* para compilar o projeto completo

usando apenas um comando. Há sistemas que usam o *MSBuild* para a compilação do código e que usam o *NAnt* para integrar outras ferramentas, como por exemplo para a execução dos testes unitários.

3.2.3.1 MSBuild

MSBuild é parte integrante da *Framework .NET* e é automaticamente integrado no VS (botão *build*). A sua configuração é definida por um documento *XML*. É limitado à *framework .NET*, tem algumas funcionalidades integradas e é ativamente desenvolvido [16]. O documento *XML* tem a definição de propriedades e os *targets* (*debug*, *release*, entre outros), onde cada *target* contém um número de tarefas [26].

3.2.3.2 NAnt

NAnt é desenhado a partir de uma ferramenta em *Java* chamada de *Apache Ant*. É uma ferramenta *open source* mantida por uma comunidade, que pode ser usada em qualquer plataforma e não é desenvolvida ativamente. É, à semelhança do *MSBuild*, controlado por um documento de configuração em *XML*. É uma boa opção para programadores que dominam *Java* e que são familiares com *Ant* [16].

3.2.3.3 Resumo

O *MSBuild* é mais adequado num ambiente *.NET*, estando integrado no VS. Como tal, foi a ferramenta escolhida. A Tabela 3 apresenta o resumo das características anteriormente especificadas.

	MSBuild	NAnt
Open source		**X**
Cross-platform		**X**
Baseado em .NET	**X**	
Configuração por ficheiro XML	**X**	**X**
Integração direta com Visual Studio	**X**	
Solução adotada	☑	

Tabela 3 – Resumo gestão de compilação

Esta solução pode, no entanto, ser complementada com o *NAnt* para execução de testes unitários em outros ambientes, ou na realização de tarefas específicas em que a sintaxe do *NAnt* melhor se adeque.

3.2.4 Utilização do TFS para testes de *software*

O TFS possui mecanismos eficazes para a automação de testes. Um dos exemplos é a ferramenta *MSTest*. Esta ferramenta permite executar testes automatizados, visualizar os resultados dos testes e guardar os mesmos tanto no disco como no TFS (ficheiro .trx[2]). No sentido da automação, e por se pretender a execução distribuída dos testes, é possível recorrer às entidades "controlador" e "agentes" de testes associados ao TFS.

O controlador de testes (*Test Controller*) é o serviço responsável por controlar a execução de testes, isto é, publica os resultados dos mesmos e coordena a sessão destes para os agentes de testes. O agente de testes (*Test Agent*) é o serviço que está sempre ligado ao controlador, está instalado numa máquina de testes e que permite a execução dos testes e recolha de informações a eles relativas [27].

O ficheiro de resultados (Figura 11) da execução de testes é analisado por parte do *tester*, e é gerado pelo VS, por tipo de teste executado.

```
<?xml version="1.0" encoding="UTF-8"?>
<TestRun id="c7636399-b674-428a-8aed-1fa457c780aa" name="Administrator@WIN2008-VM-680 2015-12-26 11:01:53" runUser=
"OPENNEBULA\Administrator" xmlns="http://microsoft.com/schemas/VisualStudio/TeamTest/2010">
  <TestSettings name="TestesDeExecucaoRemota" id="3d6f4d67-04cf-43ba-bccb-74c59259d386">
  <Times creation="2015-12-26T11:01:53.9645360+00:00" queuing="2015-12-26T11:02:01.0710540+00:00" start=
  "2015-12-26T11:02:09.4992849+00:00" finish="2015-12-26T11:30:25.7574367+00:00" />
  <ResultSummary outcome="Failed">
    <Counters total="864" executed="864" passed="841" error="0" failed="20" timeout="0" aborted="0" inconclusive="3"
    passedButRunAborted="0" notRunnable="0" notExecuted="0" disconnected="0" warning="0" completed="0" inProgress="0"
     pending="0" />
    <CollectorDataEntries>
  </ResultSummary>
  <TestDefinitions>
  <TestLists>
  <TestEntries>
  <Results>
</TestRun>
```

Figura 11 – Ficheiro de resultado de testes

[2]A execução de testes gera um ficheiro XML com os resultados dos testes, com a extensão .trx, que contém a informação sobre a execução e resultados dos testes [45].

Neste ficheiro são apresentadas informações relacionadas com os resultados dos testes, como por exemplo, o número de testes executados (864), dos quais 841 tiveram sucesso, 20 tiveram insucesso e 3 foram inconclusivos. Também é possível verificar que agente de teste executou o teste e os detalhes associados a erros ocorridos nos testes que falharam.

3.2.5 Utilização de *Cloud Computing* para CI

No livro *Cloud Computing Bible* [28], de Barrie Sosinky, o autor afirma que a "nuvem" já é utilizada há muito tempo. O termo *Cloud Computing* refere-se a tecnologia, aplicações e serviços que são executados numa rede distribuída, usando recursos virtualizados e acedidos por padrões e protocolos da Internet.

O termo "nuvem" implica a definição de dois conceitos: abstração e virtualização. Abstração devido à camada de abstração nos detalhes da implementação do sistema perante os utilizadores e os programadores, onde estes não se preocupam com *hardware*, com o armazenamento de informação ou administração de sistemas. A virtualização pelo facto da *Cloud Computing* virtualizar sistemas por gerir os recursos disponíveis, permitindo distribuir a escalabilidade, tornando a infraestrutura ágil, e também por permitir adaptar os custos às necessidades do cliente da *cloud*.

Cloud Computing pode ser dividido em conjuntos de modelos, *deployment* e serviços. Modelos de *deployment* referem-se à localização e gestão da infraestrutura, e de acordo com a definição da NIST[3], dividem-se em *cloud* privada, pública, hibrida e comunitária. A infraestrutura da *cloud* pública está disponível para a utilização ao público, pertencendo a organizações que vendem serviços *cloud*. A infraestrutura de uma *cloud* privada é exclusiva para o uso de uma organização e é gerida pela própria. A *cloud* híbrida combina múltiplos tipos de *cloud,* formando uma unidade quando as mesmas interagirem entre si. E por fim, a *cloud* comunitária é uma *cloud* que foi organizada para

[3] Instituto Nacional de Padrões e Tecnologia dos E.U.A (NIST) é uma agência governamental dentro do Departamento de Comércio dos E.U.A e tem como missão promover a inovação e competitividade industrial no país por apostar em ciência, tecnologia e padrões [41].

servir um propósito comum, sendo integrada numa ou várias organizações que partilhem preocupações comuns como missão, seguranças ou politicas.

Modelos de serviços referem-se aos serviços acessíveis da plataforma da *cloud* e são divididos por Infraestrutura como um serviço (IaaS), Plataforma como um serviço (PaaS) e Software como um serviço (SaaS).

IaaS fornece máquinas virtuais, armazenamento virtual, infraestruturas virtuais e outros componentes de *hardware* como recursos que os clientes podem utilizar. *Amazon Elastic Compute Cloud (EC2), Eucalyptus* e *GoGrid* são exemplos de IaaS.

PaaS fornece máquinas virtuais, sistemas operativos, aplicações, transações, serviços e estruturas de controlo, gerindo a infraestrutura da *cloud*. *Force.com, GoGrid CloudCenter* e *Google AppEngine* são exemplos de PaaS.

E por fim, SaaS é um ambiente de operação completo, com aplicações, gestão e interface do utilizador. Normalmente é exposto ao cliente por uma interface acedida por *browser*, possibilitando-lhe gerir as suas informações. *GoogleApps, SQL Azure* e *Oracle On Demand* são exemplos de SaaS [28].

Duas características valiosas nos serviços da *cloud* são a escalabilidade e elasticidade. Estes termos significam coisas diferentes para a *cloud*, e segundo Chip Popoviciu [29], encontram-se em momentos diferentes do ciclo de vida do negócio. A escalabilidade da *cloud* suporta necessidades estratégicas a longo prazo, ou seja, a procura dos serviços foi planeada e prevista, sem haver necessidade de grandes investimentos na infraestrutura. Já a elasticidade, suporta necessidades técnicas a curto prazo, permitindo ajustar dinamicamente os recursos consoante a procura dos serviços.

3.2.6 OpenNebula

Na estrutura do SIJ é utilizada a ferramenta OpenNebula para a execução de testes de *software*. Esta plataforma oferece uma solução simples mas flexível e rica em funcionalidades para a gestão dos recursos associados à *cloud* [30]. O OpenNebula funciona como um *wrapper*, isto é, integra qualquer componente e/ou qualquer infraestrutura, do sistema ou de outro *software*, agregando e centralizando as

operações existentes, de forma centralizada. Foi desenvolvido para que estas integrações fossem simples de realizar e utilizar. Assim, o resultado é um sistema modular que pode implementar uma variedade de arquiteturas e pode interagir com múltiplos serviços. É considerada uma Infraestrutura como Serviço (IaaS), dado que proporciona gestão de imagens e contexto, armazenamento, rede, monitorização e calendarização, virtualização, gestão de utilizadores e papéis [31]. O OpenNebula *Sunstone* é uma interface gráfica de utilizador que funciona como centro de operações do OpenNebula e é utilizada por administradores e utilizadores, simplificando a gestão de operações típicas como gestão de utilizadores, servidores ou máquinas virtuais [32].

O mecanismo de aprovisionamento de alocação de recursos do OpenNebula pode ser explicado, por exemplo, com o mecanismo de criação de uma VM. Assumindo que o *hardware* está configurado, quando há uma ação de criação de VM, o OpenNebula:

1. Verifica que existem recursos disponíveis nos anfitriões registados (disco, memória e CPU);
2. Conforme a política de agendamento (*scheduler*), a memória e CPU são alocados para a VM, no anfitrião selecionado;
3. Conforme a política de *datastore* (neste caso, *qcow shared* [33]), são criados (ou copiados) *links* para as imagens das VMs;
4. O processo do *hypervisor* (neste caso, *qemu-kvm* [34]) é executado, e tem como argumentos um ficheiro XML, os discos, memória e CPU;
5. A partir desse momento o ciclo de vida da VM passa a ser gerido pelo *hypervisor*.

Deste modo, as ações realizadas, tanto através da interface, como de API's, são traduzidas para comandos do *hypervisor*. Estes comandos, executados pelo *hypervisor*, representam as ações definidas pelos utilizadores.

Grande parte das funcionalidades do OpenNebula está disponível mediante serviços e API's, catalogadas internamente como *Cloud interfaces* e *System interfaces*. A primeira tem como propósito desenvolver ferramentas para o utilizador final, com um elevado nível de abstração das funcionalidades da *cloud.* Esta interface, ainda que mais simples e concetualmente mais próxima das necessidades neste caso de estudo, foi descontinuada. A segunda expõe todas as funcionalidades do OpenNebula e é usada

para adaptar e ajustar o comportamento do OpenNebula na infraestrutura desejada [35]. As *System interfaces* proporcionam API's de baixo nível. Estas API's de integração são a *XML-RPC* e a *OpenNebula Cloud API* (OCA). O OpenNebula oferece ainda *drivers* de interface, que permitem a interação entre o OpenNebula e a restante infraestrutura. As áreas cobertas pelos *drivers* são armazenamento, virtualização, monitorização, autorização e *networking*.

3.2.6.1 XML-RPC API

A interface *XML-RPC* é a interface primária do OpenNebula e expõe todas as funcionalidades da mesma. Esta interface é usada, por exemplo, para desenvolver bibliotecas especializadas para aplicações *cloud* ou para os casos em que o programador necessita de uma interface de baixo nível para o núcleo do OpenNebula. Permite controlar e gerir qualquer recurso da *cloud*, incluindo VMs, usando ações, como por exemplo desligar, retomar, reiniciar e arrancar VMs.

3.2.6.2 Ruby/Java OpenNebula Cloud API (OCA)

A OCA está disponível em bibliotecas *Ruby*/*Java*, e permite uma integração simples, nestas linguagens, com o núcleo do OpenNebula. É composta por um conjunto de bibliotecas que facilitam a comunicação com a interface *XML-RPC* [36]. Esta API deve ser utilizada se se está a desenvolver uma ferramenta IaaS que necessita de total acesso às funcionalidades do OpenNebula [35].

3.2.6.3 OneFlow API

A API *OneFlow* é um serviço *RESTful* para gerir, criar, controlar e monitorizar aplicações com várias camadas ou serviços compostos para VMs interligadas. Todos os dados são enviados/recebidos por *JSON* [35].

3.2.6.4 Resumo

As API's *XML-RPC* e OCA assentam na mesma estrutura, diferenciando-se na linguagem de comunicação, enquanto a *OneFlow* se baseia em pedidos *HTTP* por *JSON*. Deste modo, a API escolhida é a *XML-RPC* visto ser de simples utilização e providenciar as funcionalidades desejadas (Tabela 4).

	XML-RPC	OCA	OneFlow
Simples	**X**		**X**
Baixo nível	**X**		
Funcionalidades principais	**X**	**X**	**X**
Camada intermédia		**X**	
Serviço RESTful			**X**
Solução adotada	☑		

Tabela 4 – Resumo API

3.3 CONSIDERAÇÕES FINAIS

As ferramentas escolhidas são todas da *framework* .NET, e são as ferramentas que já são utilizadas na estrutura atual do desenvolvimento. Estas ferramentas fornecem todas as funcionalidades necessárias para a estrutura e como tal, o TFS foi escolhido para SCV e servidor de CI, o *MSBuild* como gestor de compilação. Já para executar os testes e coordenar os mesmos foi escolhido o *MSTest* e agentes e controlador de testes do VS e a *cloud* escolhida foi o OpenNebula. Relativamente à *cloud*, pretende-se tirar partido da API *XML-RPC* para acesso aos recursos da *cloud*.

4 MODELAÇÃO

A conceção de uma arquitetura envolve o estudo e modelação de componentes que são necessárias para a elaboração da mesma, bem como a análise de casos de uso, modelo de dados e diagramas de fluxos, entre outros, para melhor entender como emparelhar a estrutura pretendida. Neste capítulo é possível observar a modelação de uma arquitetura que possibilita a automatização da compilação e execução de testes, permitindo otimizar a elasticidade da *cloud*. Além da análise e modelação das várias componentes, ainda é estudada a biblioteca que facilita a comunicação entre a arquitetura e a *cloud* OpenNebula.

4.1 DESCRIÇÃO DA ARQUITETURA

Como descrito nos capítulos anteriores, a execução dos testes já estava assente numa arquitetura funcional porém que não respondia totalmente às necessidades de distribuição dos recursos da *cloud*. A estrutura da arquitetura desenhada pode ser distinguida em três fases: Configuração da compilação, processo de CI e comunicação com a *cloud*.

→ Configuração da compilação

Para suporte ao *tester* relativamente à configuração das compilações, foi desenvolvida uma aplicação *web* que se comporta como *dashboard*. A componente do diagrama que representa a configuração da compilação pode ser visto na Figura 12.

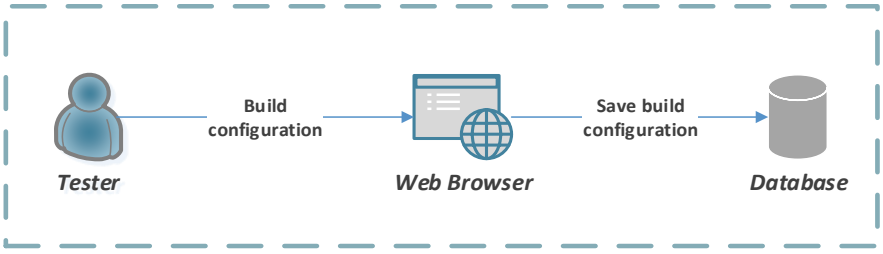

Figura 12 – Diagrama de Arquitetura - configuração de compilação

A aplicação *web* comunica com uma base de dados para guardar os dados definidos na mesma.

→ Estrutura de CI

Na Figura 10, exibida anteriormente, é apresentada a estrutura normal de CI. Há o envolvimento de programadores que trabalham sobre um ambiente de desenvolvimento, colocam as mudanças no SCV, enquanto o servidor de CI procura por mudanças nesse repositório. O servidor de CI corre um *script* que contém informações da configuração da compilação, integra a base de dados, executa os testes e faz o *deploy* de *software* para a *cloud*.

→ Comunicação com a *cloud*

O *script* é a peça essencial da arquitetura visto que consume os registos da base de dados que contêm informações da configuração da compilação (Figura 13).

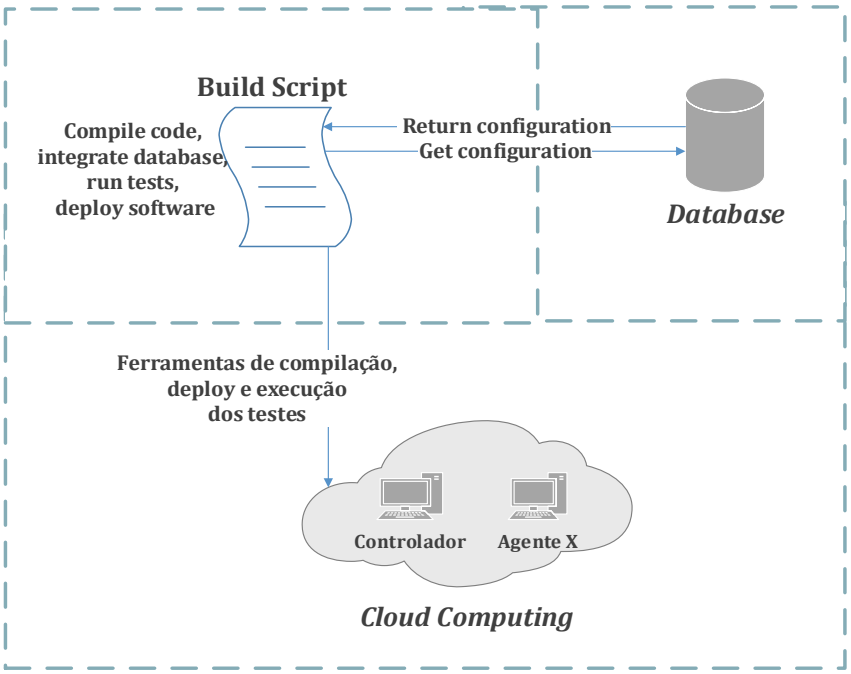

Figura 13 - Diagrama de Arquitetura – Script

Executa a compilação e faz o *deploy* do sistema, comunicando com a *cloud* para gestão das VMs (agentes de testes) que executam os testes remotamente.

4.2 CASOS DE USO

A equipa de testes tem à sua disposição uma aplicação *web* que permite gerir dados sensíveis da configuração de uma compilação automática. Neste subcapítulo são apresentados alguns dos casos de uso da aplicação *web*, bem como alguns casos de uso entre o *tester* e outras ferramentas importantes em todo o processo de compilação automática.

→ Configuração de compilação automática

O *tester* interage com a aplicação *web* para fazer a gestão de configurações gerais (Figura 14) e também para visualização de componentes importantes das compilações.

De acordo com a Figura 14, é possível descrever os seguintes casos de uso:

- **Definir informações genéricas**: O *tester* deve inserir qual é a versão do SIJ onde os testes vão executar e qual é o identificador do *template* utilizado para a criação de agentes de testes;
- **Definir tipos de configuração**: O *tester* tem a possibilidade de adicionar dois tipos de configuração, estático e dinâmico, por utilizador autorizado;
- **Definir tipos de testes**: O *tester* tem a possibilidade de adicionar tipos de testes (base de dados, interfaces, *workflows*, etc) e de selecionar quais deles irão ser executados na compilação;
- **Definir informações de VMs**: O *tester* tem a possibilidade de definir quais são as VMs (ou agentes de testes) que irão executar os testes, inserindo os dados como o identificador da mesma no OpenNebula ou o seu IP;
- **Definir utilizadores autorizados**: O *tester* deve definir que utilizadores associados ao TFS têm autorização para a execução das compilações.

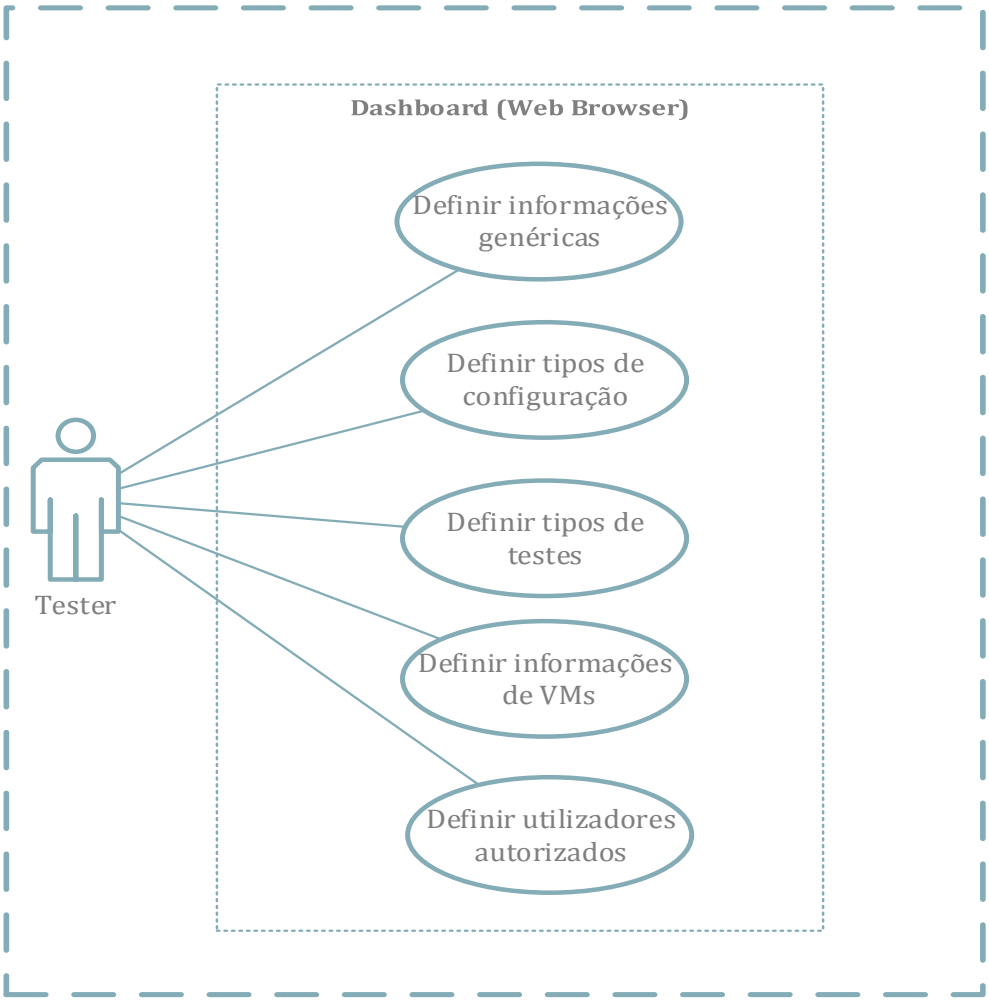

Figura 14 – Caso de uso - Dashboard

O *tester* também pode consultar detalhes de componentes importantes das compilações (Figura 15).

De acordo com a Figura 15, é possível descrever os seguintes casos de uso:

- **Visualizar compilações**: O *tester* tem a possibilidade de ver o histórico e os detalhes de cada compilação executada, e ainda verificar se há alguma ativa no momento;
- **Visualizar performance**: O *tester* tem a possibilidade de ver o histórico e os detalhes de performance de cada compilação executada, como os dados de CPU e de memória. Pode ainda consultar os dados dos mesmos no momento atual;

- **Visualizar VMs**: O *tester* tem a possibilidade de ver todos os agentes de testes associados a compilações, e os detalhes de cada um. Pode ainda consultar o estado atual dos mesmos na *cloud*;
- **Visualizar resultados de testes**: O *tester* tem a possibilidade de consultar o histórico e detalhes dos resultados de testes de cada compilação executada.

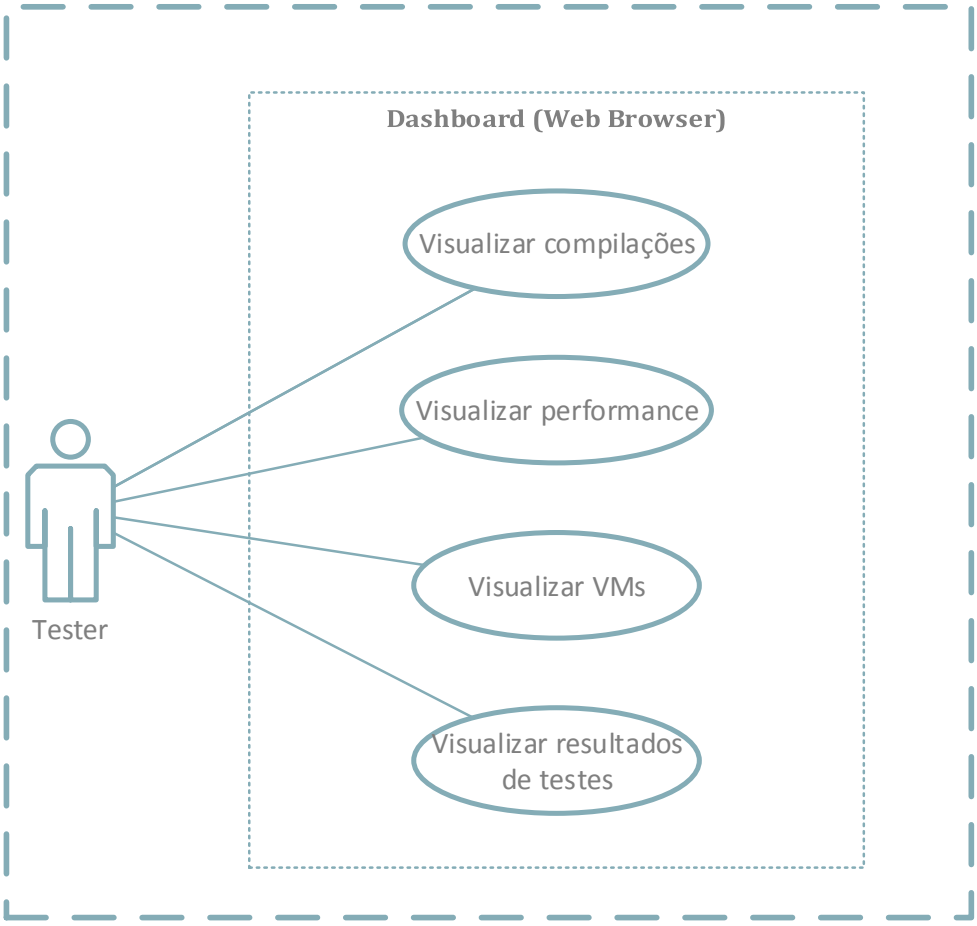

Figura 15 - Caso de uso - Dashboard

Existem ainda alguns cenários em que o *tester* tem de interagir manualmente com ferramentas de desenvolvimento: interação com VS, página *web* do *Sunstone* e também ficheiro de resultados dos testes.

→ Agendamento de compilação automática

O *tester* comunica com o VS para agendar a compilação automática, como pode ser visto na Figura 16.

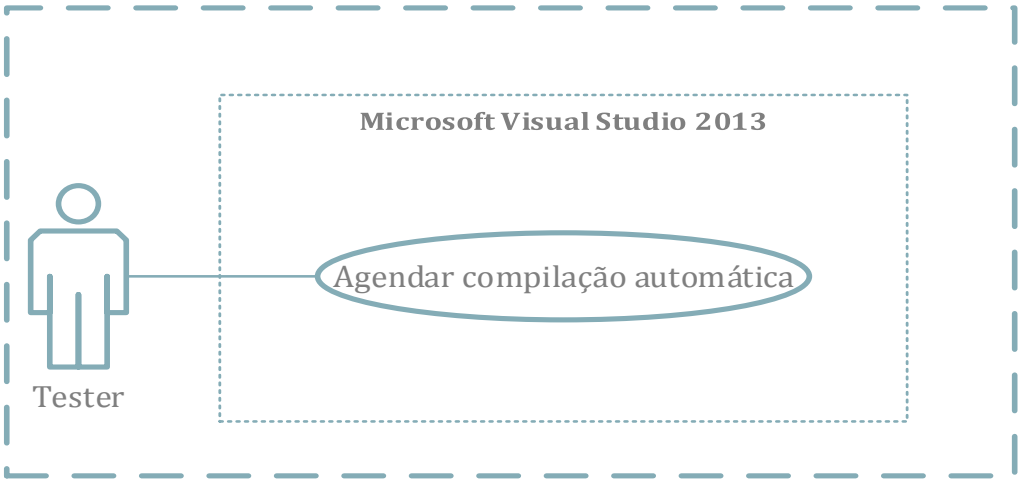

Figura 16 – Caso de uso - Agendamento de compilação automática

De acordo com a Figura 16, é possível descrever o seguinte caso de uso:

- **Agendar compilação automática**: O *tester* interage com o VS, para comunicação com o controlador de compilação, com o propósito de agendamento de uma compilação automática (tipicamente a primeira vez que a mesma vai acontecer).

Existem outros cenários de interação com o VS, como por exemplo quando for necessário reiniciar o controlador e agentes de testes (como mencionado na Figura 8).

4.3 MODELO DE DADOS

Associada a esta arquitetura foi desenvolvido um modelo de dados que auxilia toda a estrutura, permitindo guardar informações associadas a detalhes de resultados de testes, estados das VMs e dados estatísticos da *performance* do OpenNebula.

O diagrama de classes pode ser visualizado na Figura 17.

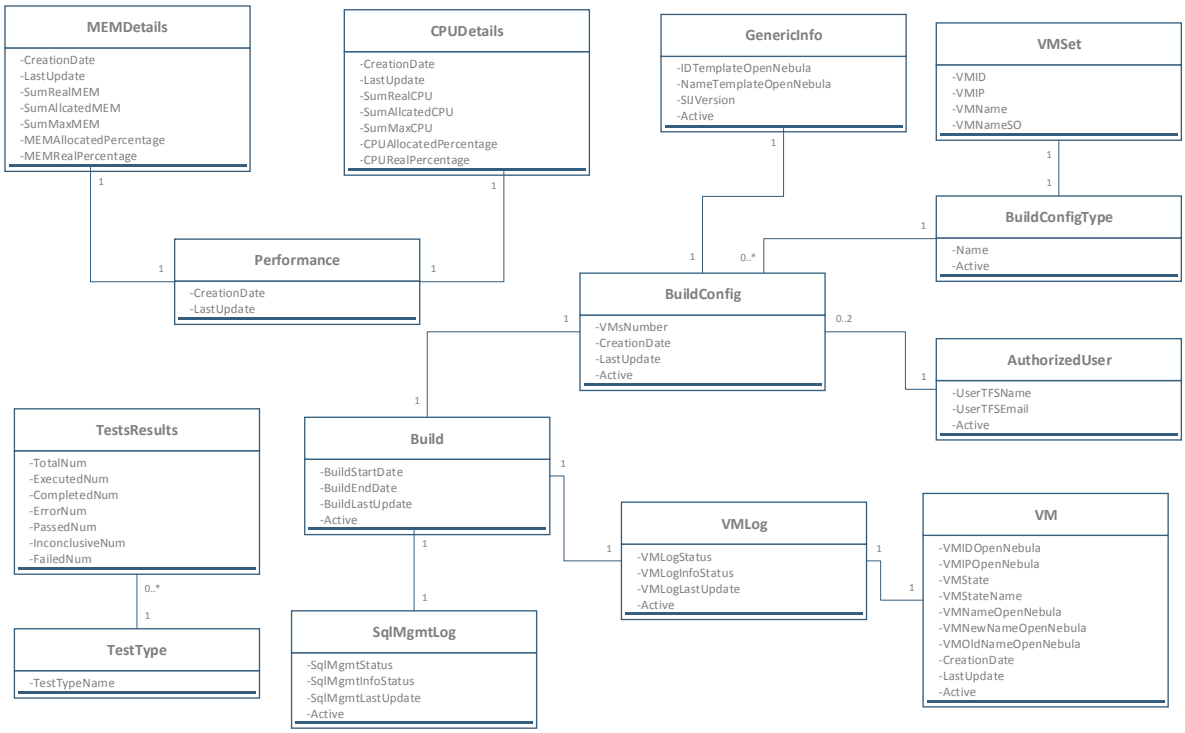

Figura 17 - Diagrama de classes

De modo a simplificar a explicação da estrutura modelada, o modelo de dados foi dividido por componentes. O modelo de dados completo pode ser visualizado em Anexo - Modelo de dados. Neste subcapítulo são abordados os dados considerados mais importantes da estrutura.

→ Configuração da compilação

Cada compilação pode estar associada a um tipo de configuração, e a este tipo está associado um conjunto de VMs, como mostra a Figura 18.

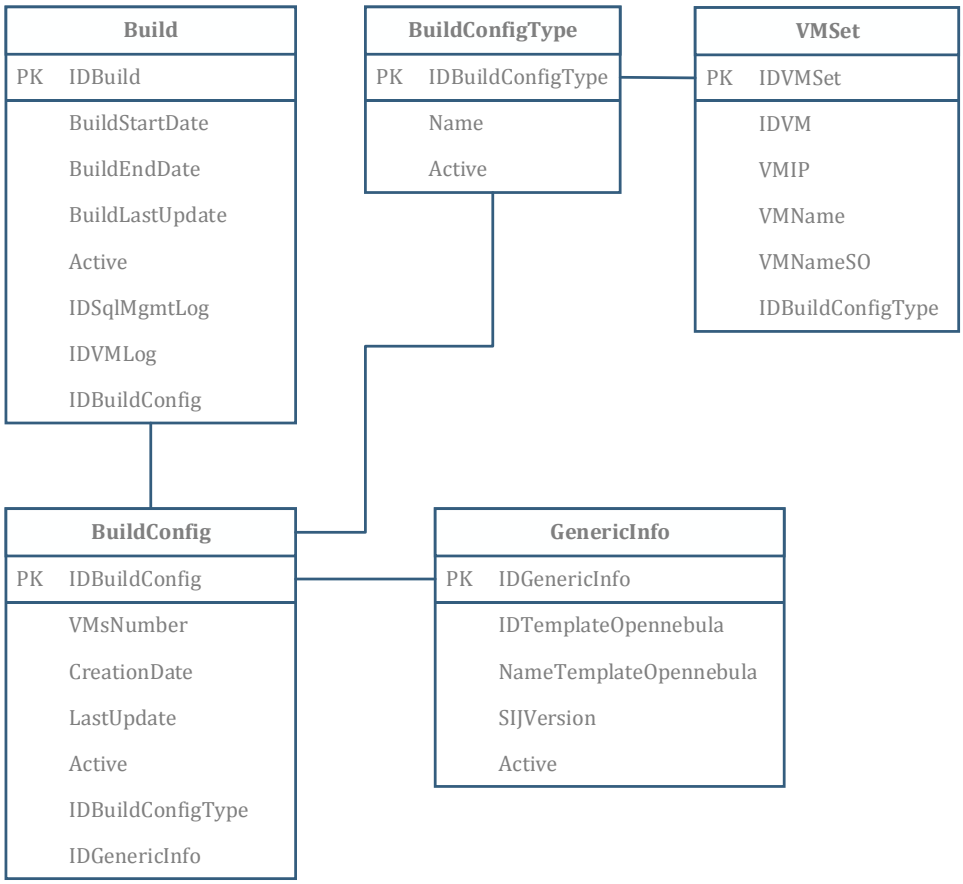

Figura 18 – Modelo de dados - Configuração da compilação

A configuração também tem associada a si informação da versão do SIJ e informação do *template* (do OpenNebula) utilizado para a criação de VMs.

→ Performance do OpenNebula

Dados da memória e processamento do OpenNebula são lidos nos momentos que o *tester* necessitar e também em cada compilação, para avaliação da estrutura desenhada. Estas informações são guardadas relativamente aos valores reais, alocados e máximos, bem como as percentagens de recursos alocados e reais. A representação encontra-se na seguinte Figura 19.

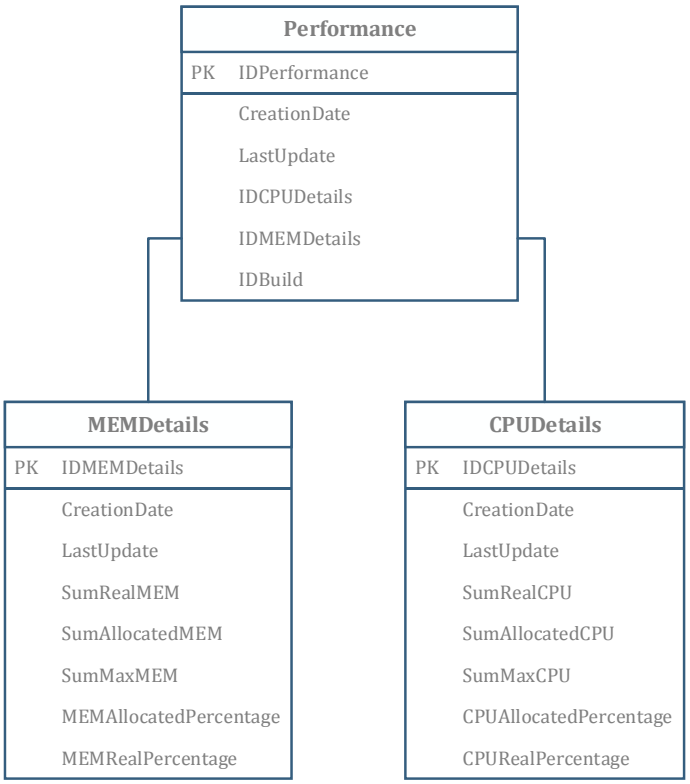

Figura 19 – Modelo de dados - Performance OpenNebula

→ Resultados da execução de testes

Para cada projeto de testes foi definido um tipo: base de dados, processo penal, processo civil, interfaces, DJE e workflows. A partir do ficheiro com os detalhes de cada execução de testes, são guardadas na base de dados as informações de quantos testes passaram, falharam, foram inconclusivos, entre outros. Assim, existe um tratamento do ficheiro para guardar estes resultados, como pode ser verificado na Figura 20.

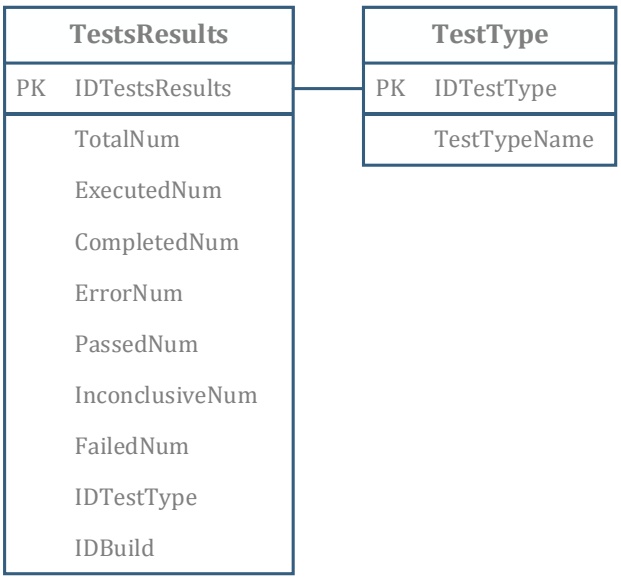

Figura 20 – Modelo de dados - Resultados de execução de testes

→ Informação das VMs

É útil guardar a informação do estado de cada VM, em cada compilação, como mostra a Figura 21.

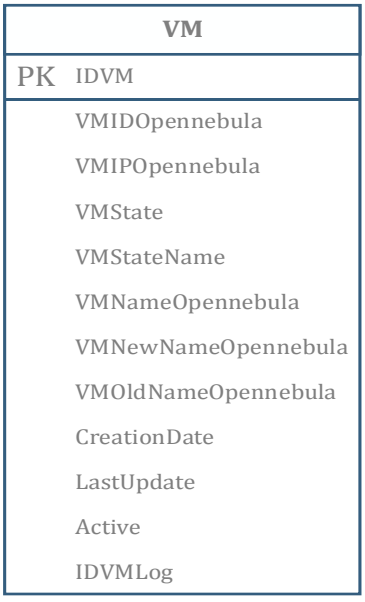

Figura 21 – Modelo de dados: Informação das VMs

Esta figura mostra as informações do nome, IP, identificador, estado (entre outros), de cada VM, e a cada uma delas está associada a uma compilação. Estes dados auxiliam na despistagem de problemas iniciais relativos à execução dos testes.

4.4 FLUXO DE COMPILAÇÃO

Para perceber o comportamento da compilação automática foi desenhado um diagrama de fluxo (Anexo - Fluxo de compilação automática) que transmite o comportamento da mesma. O fluxo da figura está dividido com duas seções.

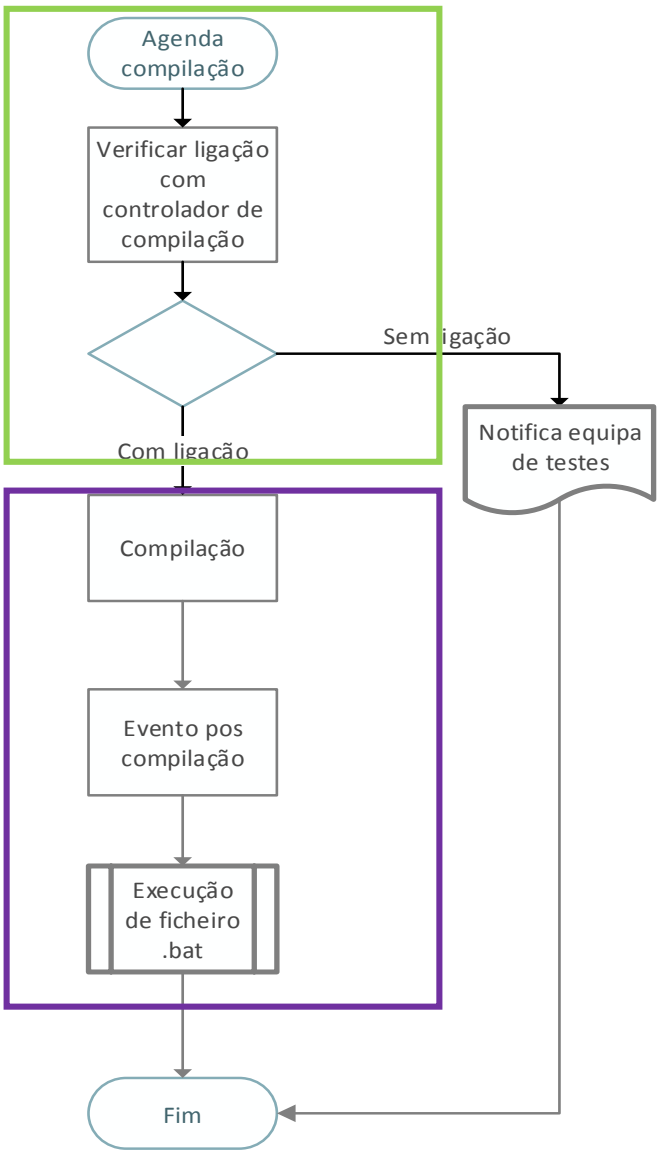

Figura 22 – Fluxo de compilação

Na componente destacada por um retângulo verde, o *tester* decide o agendamento de uma compilação automática no VS (quer seja para o momento, a cada integração, todos os dias à mesma hora, etc.), e começa por estabelecer ligação com o controlador de compilação do TFS. Neste controlador encontra-se definida a coleção do projeto de equipa (*Team Project Collection*[4]) e o projeto de equipa (*Team Project*).

Se não há ligação com o controlador de compilação, então a equipa de testes é notificada e a compilação automática é cancelada.

Por outro lado, quando há ligação com o controlador de compilação, o processo segue para uma fase onde todos os projetos dentro da solução do SIJ são compilados, representado pela componente destacada no retângulo roxo. Se as compilações forem realizadas com sucesso, é invocado um evento de pós-compilação que irá executar um subprocesso. O subprocesso invocará um ficheiro de linha de comandos (extensão .bat).

A definição do evento de pós-compilação (executado quando a compilação for bem sucedida) é definido como mostra a Figura 23.

```
<PropertyGroup>
  <PostBuildEvent>call C:\Folder\file.bat</PostBuildEvent>
</PropertyGroup>
```

Figura 23 – Evento pós-compilação

Neste ficheiro de linha de comando (.bat) é executado um serviço de configuração que faz toda a gestão das VMs e da execução dos testes, que será detalhadamente explicado no próximo capítulo. Quando este subprocesso termina, a compilação automática também termina.

[4] Um projeto (*Team Project* [42]) é um repositório de código fonte e uma ferramenta que permite que os membros do projeto façam o planeamento, verifiquem os progressos do desenvolvimento e colaborem entre eles. Um projeto insere-se dentro de uma coleção de projetos (*Team Project Collection* [44]) e esta coleção é integrada na arquitetura lógica do TFS.

4.5 BIBLIOTECA XML-RPC.NET DE CHARLES COOK

Todas as componentes da arquitetura de CI foram estudadas no subcapítulo Ferramentas para CI, porém é necessário entender como é que a estrutura de CI comunica com o OpenNebula através da API *XML-RPC*. A estrutura de CI estará assente sobre tecnologias *.NET* e assim é necessário uma camada intermédia que torne possível a comunicação com a *cloud*.

Em março de 2001 foi desenvolvida uma biblioteca que permite a implementação de serviços *XML-RPC* [37] no ambiente *.NET*. Scott Hanselman escreve que o autor Charles Cook criou uma biblioteca elegante chamada de XML-RPC.NET e disponibilizou-a à comunidade. Algumas das funcionalidades da biblioteca são: interface baseada na definição de servidores e clientes *XML-RPC*, geração de código de *proxies* seguros do cliente, serviços *web* em *ASP.NET* com suporte a SOAP [5] e *XML-RPC*, suporte de chamadas assíncronas ao cliente e suporte de ficheiros *XML* codificados, suporte no lado do servidor à API *XML-RPC*, entre outras funcionalidades.

Para realizar a comunicação com a API XML-RPC.NET, basta aceder a uma das interfaces que a biblioteca de Charles Cook disponibiliza (ou criar uma interface), escolher os métodos a que se pretende aceder e definir qual é o endereço do *XML-RPC* da *cloud*.

Na Figura 24 é possível ver a interface que permite estabelecer o endereço do *XML-RPC* do OpenNebula, e o método que permite comunicar com cada VM. O método em causa permite alocar uma nova VM na *cloud*. Na mesma figura é possível verificar que a biblioteca de Charles Cook dispõe de informações acerca do método que será invocado (descrição).

[5] Protocolo de acesso a objetos simples (SOAP) é um protocolo de comunicação com a aplicação, baseado em *XML*, fornecendo uma maneira de comunicação entre aplicações que executam em tecnologias ou linguagens de programação diferentes, fazendo o envio e receção de mensagens [50].

```
namespace OpenNebulaHosting.OpenNebulaAdapter
{
   [XmlRpcUrl("http://cloudpt2.clients.ua.pt:2633/RPC2")]
   public interface XmlRpcVirtualMachineManagement : IXmlRpcProxy
   {

      [XmlRpcMethod("one.vm.allocate")]
      Array oneVirtualMachineAllocate(string sessionSHA, string
atributeValueTemplate);

      /*Description: Allocates a new virtual machine in OpenNebula.

         IN       String  The session string.
         IN       String  A string containing the template for the vm. Syntax can be the
usual “attribute=value” or XML.
         OUT      Boolean        true or false whenever is successful or not
         OUT      Int/String     The allocated resource ID / The error string.
         OUT      Int      Error code.*/

   }
}
```

Figura 24 – Cliente para acesso a VM da cloud

Na camada lógica do código é invocado o método desta mesma interface, e a informação devolvida pelo OpenNebula é processada conforme o programador desejar. O número de parâmetros enviados para cada método disponível na *XML-RPC* pode ser diferente, porém a resposta de todos os métodos tem uma estrutura comum: uma variável binária, que identifica se a resposta ao método invocado ocorreu com sucesso ou incesso, uma mensagem com a resposta do método (ou em caso de erro, uma mensagem a identificar o mesmo) e por fim um valor inteiro que identifica o código de erro (se este existir, senão o valor retornado é 0).

Os métodos do OpenNebula utilizados para esta arquitetura encontram-se na Tabela 5 exposta de seguida.

Método	Significado	Parâmetros
one.vm.info	Recuperar informação de uma VM;	2
one.vm.monitoring	Retorna os registos da monitorização da VM;	2
one.vm.action	Submete uma ação a ser efetuada em uma VM. Ações: *resume*, *poweroff*, *delete*, *undeployed*;	3
one.template.instantiate	Cria uma nova VM a partir de um template;	3
one.host.info	Recupera informação de todos os *hosts.*	1

Tabela 5 – Métodos OpenNebula utilizados

4.6 CONSIDERAÇÕES FINAIS

A modelação da arquitetura proposta foi detalhada neste capítulo, apoiando-se na exposição de diagramas de arquitetura, casos de uso, modelo de dados e fluxo de compilação.

5 IMPLEMENTAÇÃO

Este capítulo foca-se em detalhes de implementação da arquitetura. Primeiramente será descrita a arquitetura de CI proposta para o SIJ, e posteriormente serão analisados os diagramas de sequência que ajudam a entender os procedimentos realizados no serviço de configuração. Este serviço, nomeado nos capítulos anteriores por "*script*", tem a intenção de fazer toda a gestão das VMs e execução dos testes. Por fim será explicada a *dashboard* para auxílio ao *tester* relativamente a configurações de compilação automática.

5.1 ARQUITETURA DE CI PARA O SIJ

Relembrando o que foi descrito nos capítulos anteriores, uma arquitetura de CI necessita de componentes-chave para o seu correto funcionamento. Todas as componentes necessárias e previamente escolhidas foram detalhadas no capítulo Estado de Arte e como tal, o ambiente de desenvolvimento é o VS 2013 com sistema de gestão de base de dados o *SQL Server* 2012. O SCV (repositório) é o TFS, e este também se comportará como servidor de CI (Figura 25).

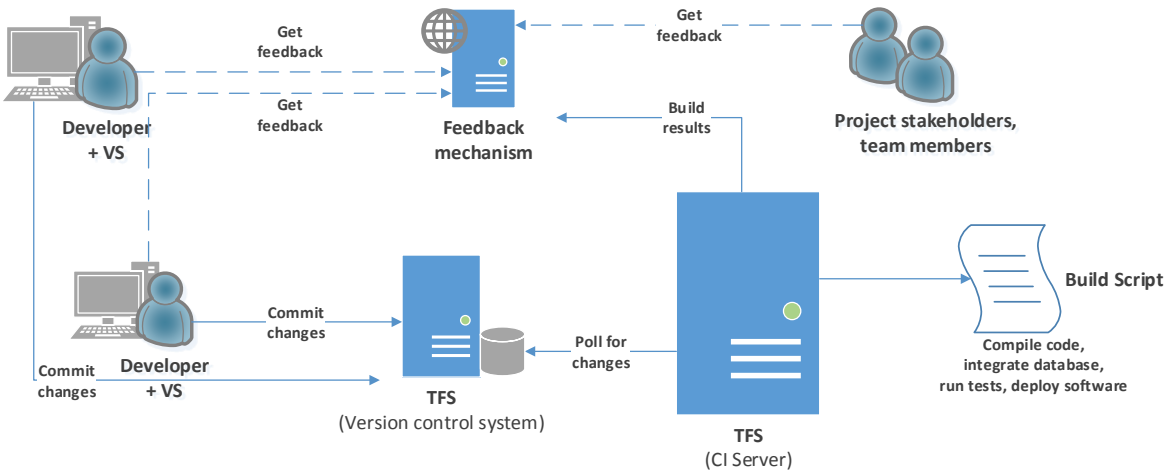

Figura 25 – Arquitetura de CI para o SIJ - base

O representado na Figura 25 é considerado uma concretização da arquitetura ilustrada na Figura 10. Na Figura 25 é possível visualizar as peças escolhidas para implementação da arquitetura de CI. Para automação de compilação será utilizada a ferramenta *MSBuild*, para o *deploy* da aplicação será utilizado o *MSDeploy* e o *MSTest* para execução de testes. Estas últimas três ferramentas irão comunicar com as VMs do OpenNebula, onde se encontram instalados o controlador e os agentes de testes. Esta componente pode ser visualizada na Figura 26.

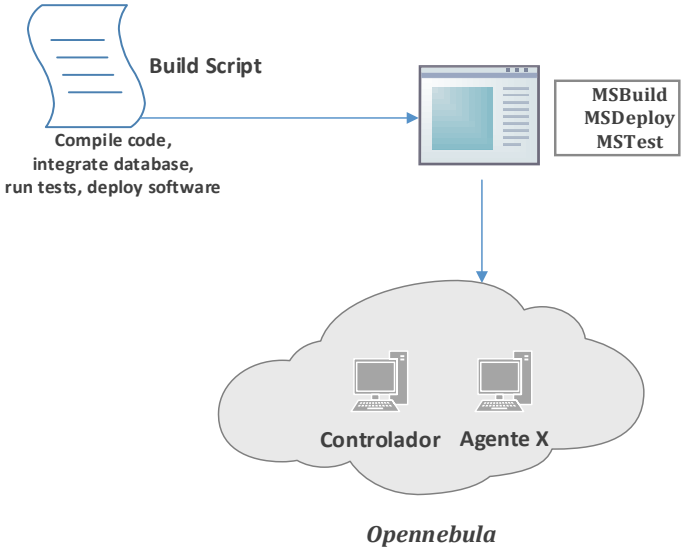

Figura 26 – Arquitetura de CI para SIJ - comunicação com cloud

O diagrama integral encontra-se no Anexo - Diagrama de arquitetura de CI para o SIJ.

5.2 APLICAÇÃO WEB - DASHBOARD

O *tester* pode optar por definir as VMs que atuam como agentes de testes manualmente ou pode optar por defini-las na aplicação *web*. A configuração manual implica a edição de um ficheiro auxiliar que se encontra na solução de desenvolvimento que será lido apenas se não houver nada inserido na base de dados relativamente às VMs. Se o *tester* optar pela interface *web,* consegue uma visão mais clara e simples dos campos a configurar. Pelo *browser* é possível realizar ações simples de definição e visualização de dados. A interface foi desenhada para ser prática e simples, servindo de suporte da

compilação automática, definida pelo *tester*, e foi desenvolvida na *framework* de desenvolvimento *MVC*[6] 5 (*ASP.NET*) com utilização de *Bootstrap*[7] para tratamento da interface de utilizador. Esta aplicação *web* comunica com o OpenNebula para leitura do estado de cada VM e dados de desempenho da *cloud*, e comunica com uma base de dados *SQL Server*, onde estão guardadas todas as informações necessárias e na qual vão ser inseridas as informações que o utilizador pretender. Apesar da apresentação da Dashboard descrita neste subcapítulo, algumas operações foram ocultadas por se considerar que os casos mais importantes foram especificados.

→ Página inicial

A página inicial da Dashboard tem a informação da última compilação executada bem como a compilação atual (se estiver a ocorrer alguma no momento) (Figura 27).

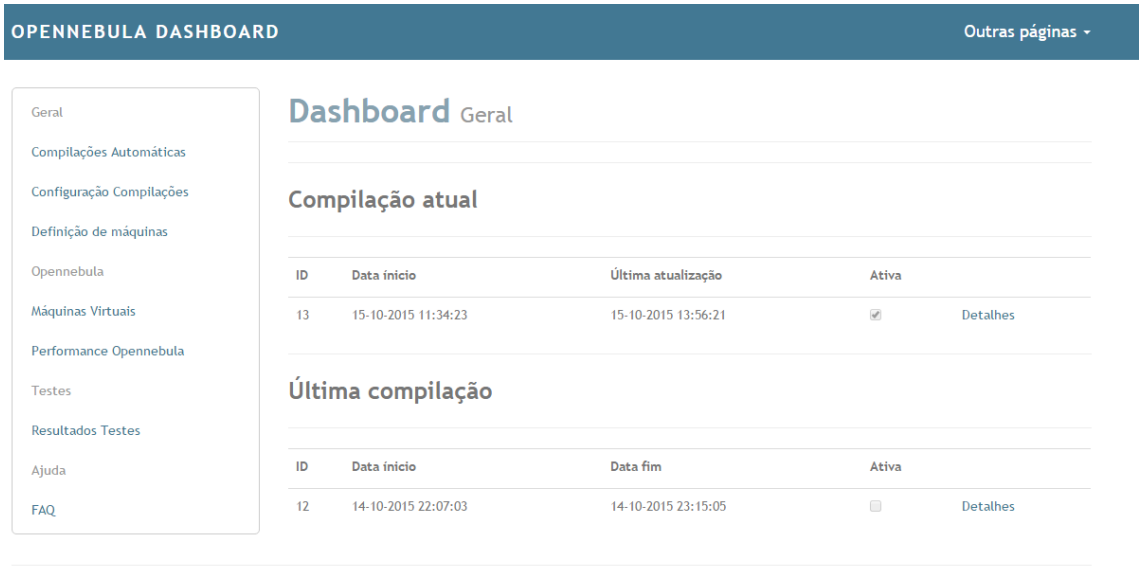

Figura 27 – Dashboard - index

[6] O modelo de programação Model-View-Controller (MVC) é um modelo desenvolvido em ASP.NET [48].
[7] Bootstrap é uma *framework* livre para desenvolvimento *web,* mais fácil e mais rápida, que inclui *templates* baseados em HTML e CSS, para vários tipos de objetos das páginas.

→ Compilações automáticas

O *tester* tem a possibilidade de visualizar as últimas compilações automáticas executadas (ou se existe alguma ativa) e os detalhes das mesmas (tipo de configuração, estado os serviços SQL). Na Figura 28 está o detalhe da compilação ativa.

Figura 28 – Dashboard - Compilações automáticas

→ Definição VMs

A definição das VMs permite ao *tester* ter a flexibilidade de definir quais são as VMs que vão atuar como agentes de testes para cada configuração. No caso da Figura 29, existem VMs para uma configuração estática, e como tal as VMs já têm os dados mais detalhados do OpenNebula. No caso de uma configuração dinâmica, o utilizador insere os nomes que pretender para identificar cada VM.

Figura 29 – Dashboard - Definição VMs

→ VMs no OpenNebula

O *tester* pode verificar o estado das VMs (registadas para a configuração estática ativa) para verificar se estão ligadas ou não (Figura 30). Relativamente à configuração dinâmica, as VMs aparecem quando as VMs forem criadas na *cloud*.

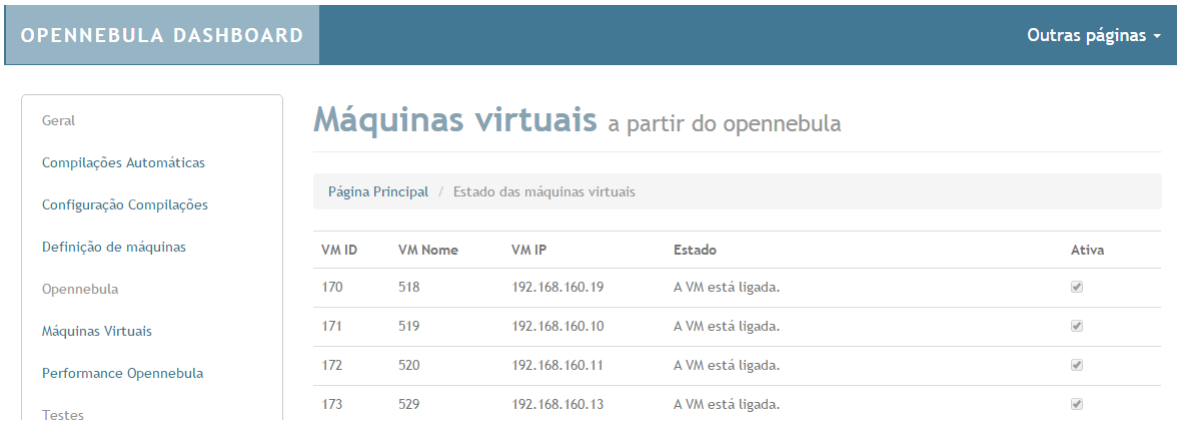

Figura 30 – Dashboard - Estado VMs

→ Performance OpenNebula

O *tester* consegue visualizar os dados de *performance* do OpenNebula nos momentos iniciais e finais da compilação automática ou quando assim o desejar. Na Figura 31 está representada uma medição realizada pelo tester num momento em que não existia nenhuma compilação ativa.

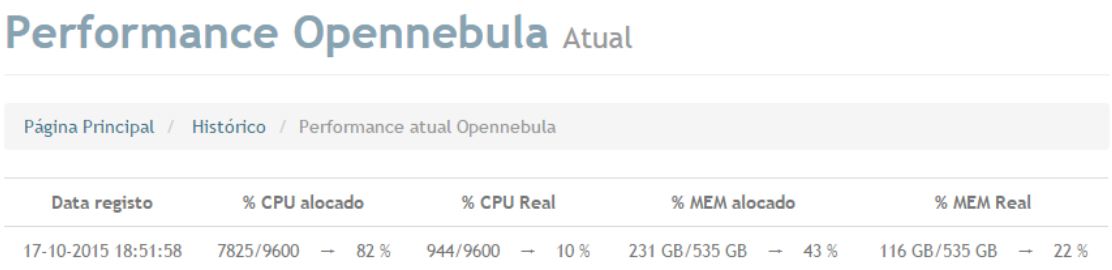

Figura 31 – Dashboard - Performance OpenNebula

→ Tipo de configuração de compilação

O *tester* tem a possibilidade de ver os dados associados à configuração da compilação automática ativa, como por exemplo o número de VMs associadas a essa configuração.

→ Utilizadores autorizados

A partir do acesso ao TFS, o *tester* pode verificar que utilizadores estão envolvidos no projeto do SIJ, e autorizar os mesmos para o lançamento de uma compilação automática. Assim, quando um membro da equipa desejar iniciar a compilação, é efetuada uma validação relativamente ao mesmo ter permissões para realização da ação desejada.

5.3 ESTRUTURAÇÃO DO SERVIÇO DE CONFIGURAÇÃO

Neste subcapítulo é analisado o serviço que permite realizar as configurações necessárias para a automação do processo de execução de testes do sistema SIJ.

O serviço de configuração tem várias funções, como a comunicação com as VMs, comunicação com a base de dados, *deploy* de todos os ficheiros necessários para execução remota dos testes pelas mesmas máquinas, execução dos testes remotos e comunicação de *feedback* ao *tester*.

O serviço está estruturado em módulos, onde cada um cumpre um papel distinto. Existe um módulo que gere as configurações da compilação, um módulo que tem todos os métodos disponíveis para a gestão das VMs (instanciação, detalhes e ações), um módulo que faz a gestão das configurações do SO de cada VM alojada na *cloud*, um módulo que gere os processos referentes ao agente de testes de cada VM, um módulo que gere os serviços SQL de cada VM, e por fim um módulo que faz a gestão do TFS para execução de testes e divulgação dos resultados. O diagrama de sequência da comunicação entre os módulos pode ser visto na Figura 32.

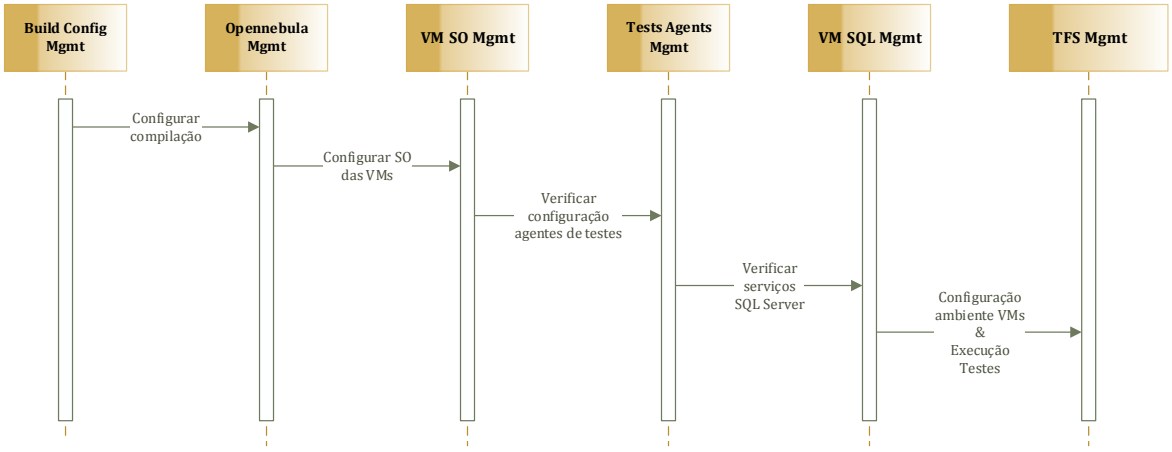

Figura 32 – Diagrama sequência - Comunicação entre módulos

O módulo de configuração de compilação gere os dados da compilação e comunica com o módulo de gestão do OpenNebula, onde são criadas ou recuperadas as VMs, sendo realizada uma verificação relativa ao estado das mesmas (se estão ou não em execução). Este módulo comunica com o módulo da gestão dos SOs das VMs para configuração das mesmas (inserção no domínio de rede ou configuração de *firewalls*). Quando o módulo de gestão dos SOs termina, comunica com o módulo dos processos de agentes de testes para validar a situação dos mesmos. Quando estes processos são validados, dá-se comunicação com o módulo dos serviços SQL para validar se estes estão em execução, e por fim, é comunicado ao módulo de gestão do TFS que pode preparar os ambientes das VMs, executar os testes e reportar os resultados. No fim, as VMs são desligadas (ou eliminadas) e o processo termina.

Posto isto, é realizada a análise a cada um dos módulos.

5.3.1 Configuração do módulo de configuração da compilação

A primeira validação realizada é ao controlador de testes. É verificada se esta VM está com ligação à rede, com a realização de um *ping*[8]. Se isto não acontecer, então a

[8] *Ping* é a ferramenta primário para resolução de problemas de conectividade a nível do IP, executado na linha de comandos do *Windows* [47].

execução é cancelada e o *tester* é reportado sobre a situação, visto que é um problema que tem de ser resolvido manualmente.

Se a VM do controlador de testes responder com sucesso ao *ping*, então é criado um registo da compilação na base de dados, onde são guardados os dados relativos ao identificador da compilação e também a data de início da mesma. Posto isto, é realizada uma leitura aos dados de memória e CPU da *cloud* para guardar registos relacionados com a *performance*. É ainda verificada qual é o tipo de configuração de compilação para determinar que sequência de passos deve ser seguida pelo módulo seguinte (gestão do OpenNebula). Considerando um cenário de configuração estática, em que os detalhes das VMs já foram inseridos pelo módulo, é realizado um mapeamento dos dados das mesmas para ficarem associadas à compilação que está em execução. A representação da sequência está na Figura 33.

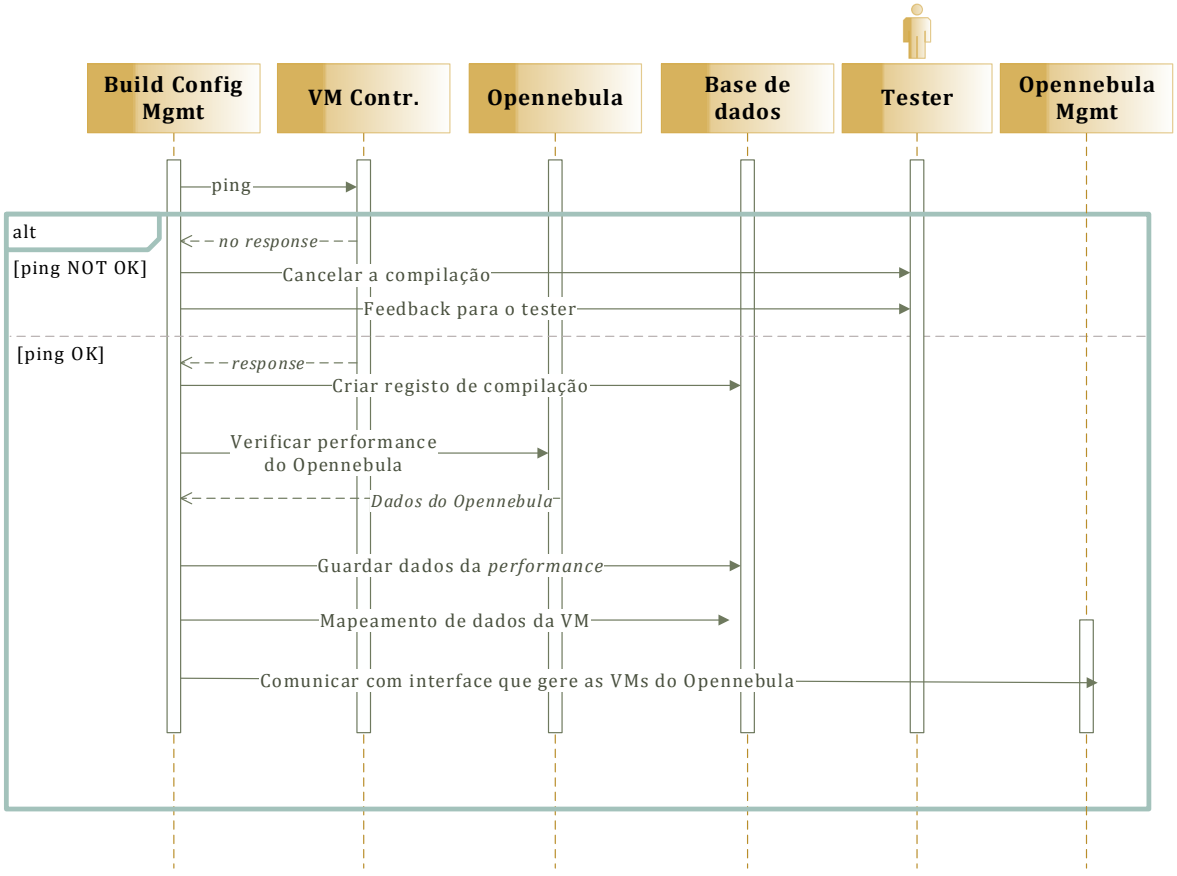

Figura 33 – Diagrama sequência: Iniciar compilação automática

5.3.2 Configuração do módulo de gestão do OpenNebula

Este módulo é responsável pela gestão de VMs e inicialmente é verificada se a compilação é estática ou dinâmica. Caso se trate uma configuração estática, as VMs são reiniciadas (para garantir um estado inicial conhecido). Se a configuração é dinâmica, são criadas as VMs definidas previamente pelo *tester*. Estas VMs têm exatamente as mesmas características (na *cloud* e nos SO's) que as VMs estáticas. A representação da criação de uma VM pode ser visualizada na Figura 34.

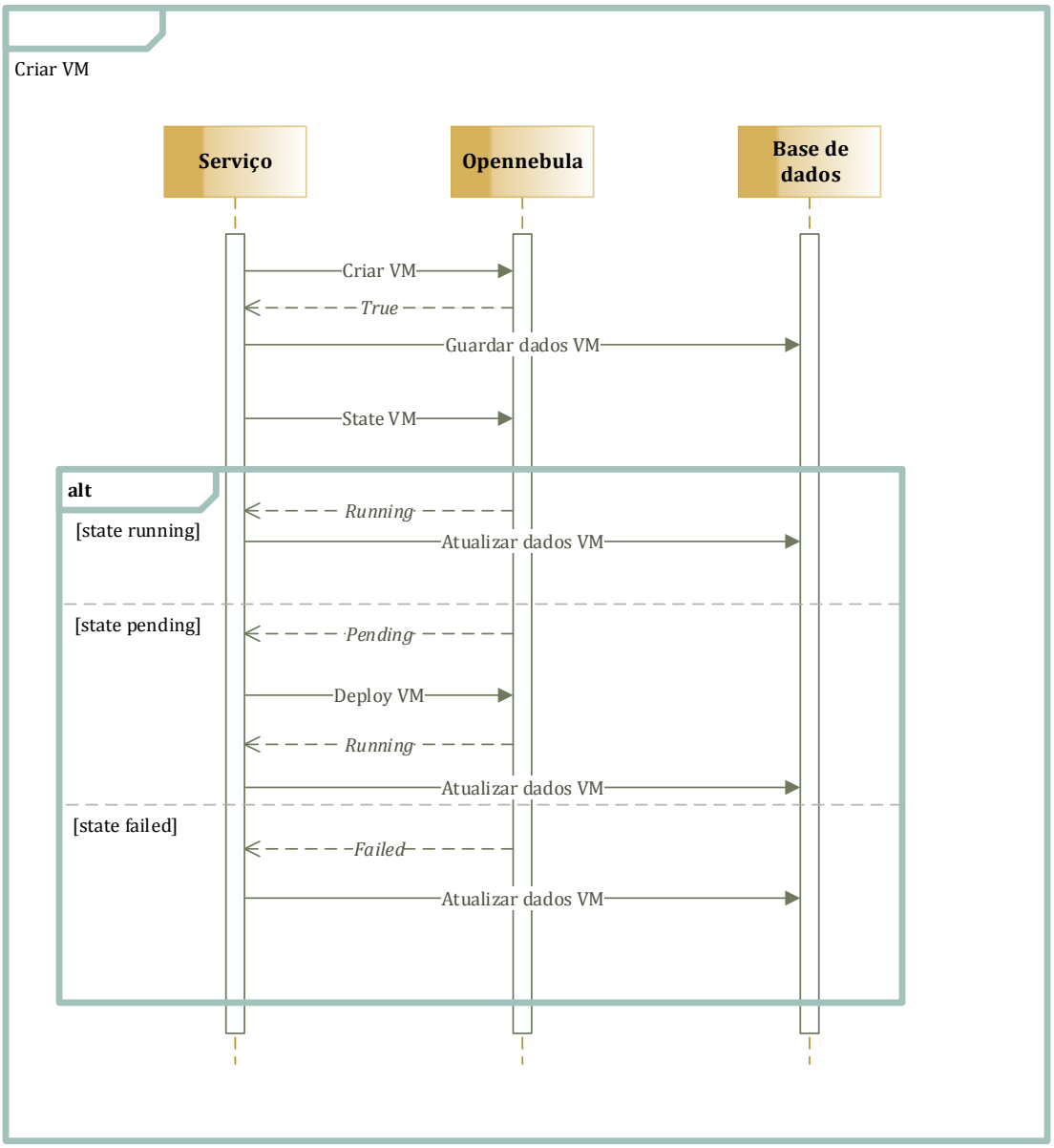

Figura 34 – Diagrama sequência - Criação de VM

Neste diagrama é possível visualizar que há diversas comunicações com a base de dados e os serviços do OpenNebula. Depois da criação da VM é realizada uma atualização na base de dados. Posteriormente é estabelecido um pedido ao estado de cada VM para verificar se é necessário fazer o *deploy* da VM no OpenNebula (de modo a ficar configurável e com IP atribuído) e de acordo com os dados retornados, os mesmos são guardados na base de dados.

Após a criação de todas as VMs, o processo segue para uma etapa onde é validado se as VMs foram criadas (ou reiniciadas) com sucesso (o diagrama completo encontra-se no Anexo - Diagrama sequência: Módulo de gestão do OpenNebula). Por vezes a publicação das VMs gera erros internos na *cloud* e estas não são criadas com sucesso. Deste modo, esta validação é essencial para a decisão sobre o próximo passo a ser executado pelo serviço. Se todas as VMs falharam na sua criação, o processo termina e o *tester* é notificado desse mesmo problema. Senão, o processo continua e é estabelecida a comunicação com o módulo que faz a gestão dos sistemas operativos das VMs de agentes de testes.

5.3.3 Configuração do módulo de gestão do SO de cada VM

Nesta seção são descritas configurações relacionadas com o registo das VMs no domínio do OpenNebula. Inicialmente é invocada uma biblioteca que simula o comportamento da ferramenta NsLookup[9], para obter o endereço IP de cada VM. Se não for encontrada informação sobre o IP, é realizado um pedido à *cloud* para devolver informações sobre a VM. Estas informações são devolvidas no formato *XML*, onde se encontram dados relacionados com o template da VM. Destes valores é possível obter o IP associado à mesma.

Se a configuração for dinâmica, então o procedimento passa por modificar o nome de cada VM e reiniciar a mesma, para que seja possível adicioná-la ao domínio. Assim que as VMs se encontram prontas, são atualizados os DNS's, para pertencerem ao domínio

[9] O Nslookup.exe é uma ferramenta administrativa de linha de comandos para testar e fazer resolução de problemas de e de DNS [46].

do OpenNebula, e as *firewall's* são desligadas para ser possível a comunicação entre o gestor do domínio e cada uma das VMs. Assim que se encontram na rede, existe uma verificação no controlador de domínio se as VMs estão registadas na AD. Se sim, são removidas. Posteriormente, as VMs são adicionadas à AD, estando finalmente sobre o mesmo domínio. As ligações de ambiente remoto (para cada VM) são iniciadas, devido ao facto da execução de testes de interface necessitar de acesso ao *desktop* de cada VM.

O diagrama de sequência deste módulo encontra-se no Anexo - Diagrama sequência: Módulo de gestão de SO das VMs. Quando este módulo realiza todos os seus comandos, é estabelecida a comunicação com o módulo que faz a gestão dos processos de agentes de testes.

5.3.4 Configuração do módulo de gestão dos processos de agente de testes

Para ser possível a comunicação entre o controlador de testes e os agentes de testes é necessário garantir que os processos dos agentes estão em execução. Assim, este módulo está encarregue de fazer essa mesma validação, e se se verificar que não estão ativos, então inicia os mesmos. Estes serviços são denominados: *QTAgentProcessUI*.exe, *QTAgentService.exe* e *QTDCAgent32.exe*.

O diagrama correspondente a este módulo encontra-se no Anexo - Diagrama sequência: Módulo de gestão de processos de agente de testes. No final é realizada a comunicação com o módulo que gere os serviços SQL de cada VM.

5.3.5 Configuração do módulo de gestão dos serviços de SQL

À semelhança do comportamento do módulo descrito anteriormente, neste módulo são verificados os serviços do SQL, e se não se encontram em execução, são iniciados. Os serviços SQL verificados neste procedimento são: *MSSQL$SQLEXPRESS2K8FTS* e *SQLAgent$SQLEXPRESS2K8FTS*.

Há uma validação relativa à configuração estática/dinâmica. Se o procedimento é relativo à configuração dinâmica, então é definido que o utilizador responsável pelo

serviço de SQL é o administrador da rede de domínio. Isto acontece para cada VM que atua com agente de testes.

O diagrama correspondente a este módulo encontra-se no Anexo - Diagrama sequência: Módulo de gestão de serviços SQL. Após esta análise, é realizada a comunicação com o módulo que faz a gestão do TFS e execução dos testes.

5.3.6 Configuração do módulo de gestão do TFS e execução de testes

Este módulo faz toda a gestão sobre o repositório de dados, onde a comunicação com o mesmo está definida em ficheiros *Powershell*. Assim, este módulo trata de executar cinco ficheiros *Powershell*:

1. Obtenção da última versão da base de dados do repositório;
2. Publicação dos ficheiros de base de dados por cada VM;
3. Obtenção da última versão do código associado à configuração de páginas web;
4. Publicação no servidor IIS em cada VM;
5. Obtenção da última versão dos ficheiros de configuração dos serviços (*MJCVService*, *Queue*, *MJCVWorkflow*) que operam em cada VM, e reiniciar os mesmos serviços.

É verificado o tipo de configuração, e se for dinâmica, são executados dois comandos adicionais:

1. Criação dos serviços de apoio ao funcionamento do sistema;
2. Partilha das pastas onde se encontram os ficheiros de configuração dos serviços.

Um exemplo da chamada do ficheiro *Powershell* está representado na Figura 35, onde é pesquisada a lista de VMs a partir da configuração, e o resultado dessa pesquisa é enviado como parâmetro do ficheiro *Powershell*.

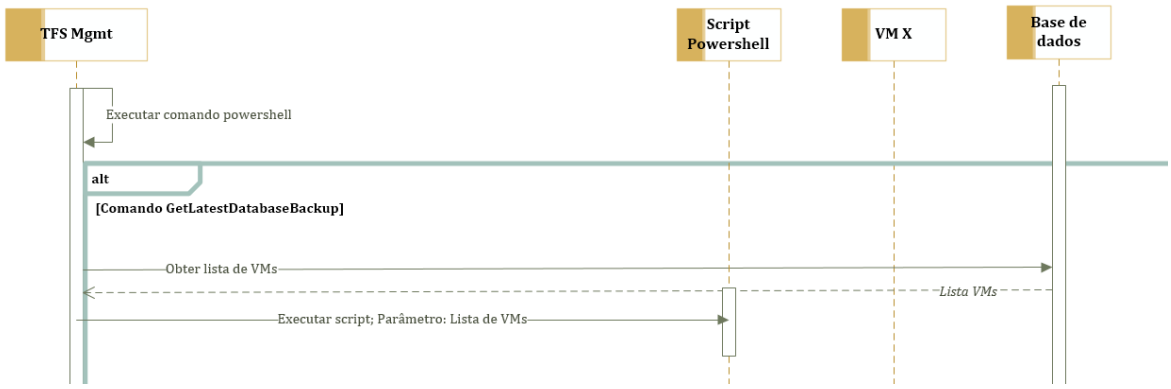

Figura 35 – Diagrama sequência - Executar ficheiro Powershell

Após a execução de todos estes ficheiros, é procedida a iniciação da execução dos testes de *software*, onde o diagrama é representado na Figura 36.

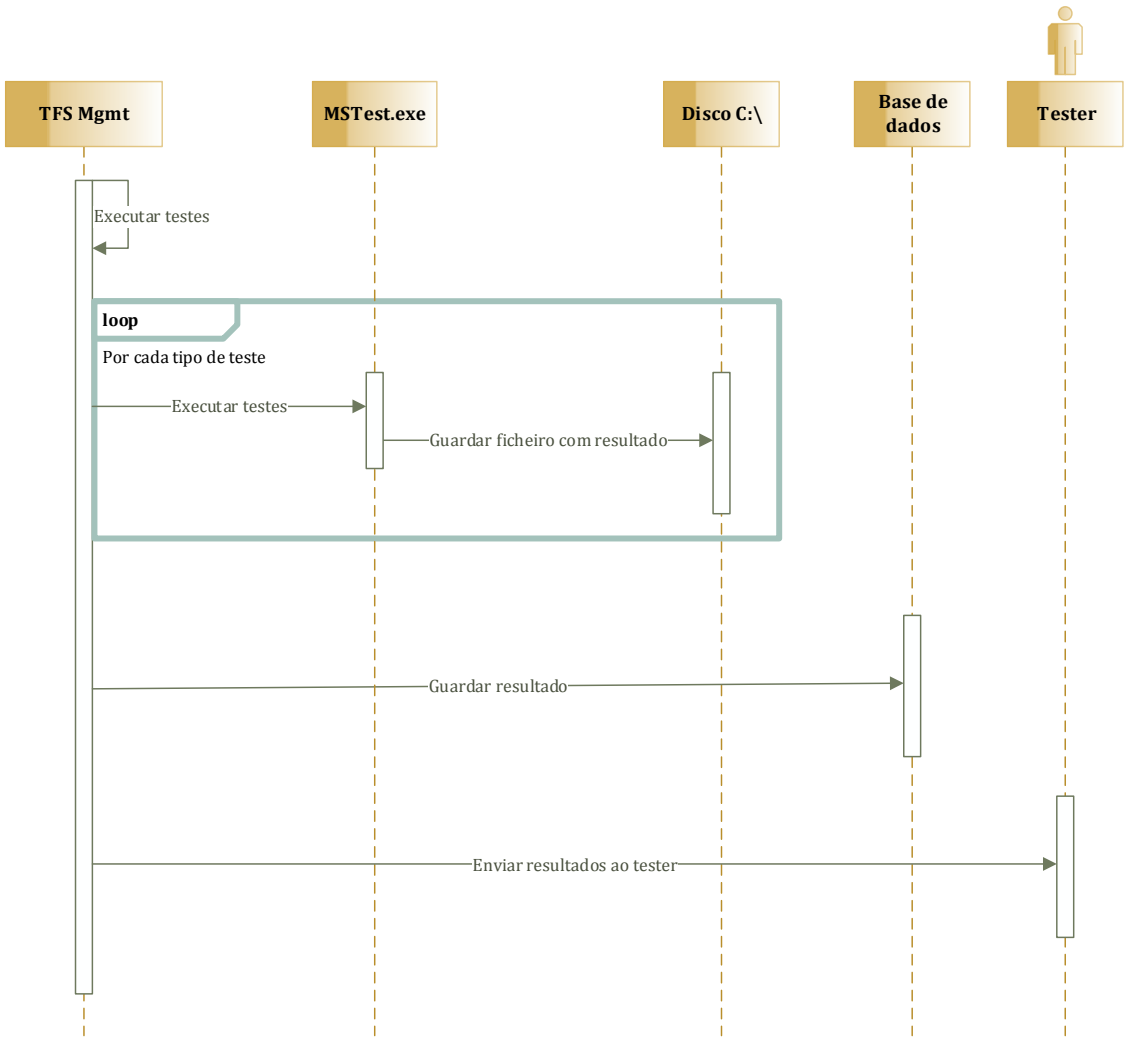

Figura 36 – Diagrama sequência - Executar testes

Os testes são executados pela ferramenta *MSTest*, especificando qual é a coleção de testes a executar e onde publicar os resultados dos mesmos. No fim da execução dos testes, o ficheiro com os resultados é analisado para extração dos valores obtidos como quantos testes passaram, falharam, inconclusivos, etc. No fim deste processo, os resultados são enviados por correio eletrónico ao *tester*, permitindo ao mesmo ter uma perceção imediata do panorama dos resultados dos testes executados. No fim de reportar os resultados para o *tester*, este módulo comunica com o módulo que faz a gestão do OpenNebula para eliminar (ou desligar) as VMs que executaram os testes na compilação realizada. Toda a sequência explicada neste subcapítulo está ilustrada com mais detalhe em Anexo - Diagrama sequência: Módulo de gestão do TFS e execução de testes.

5.4 CONSIDERAÇÕES FINAIS

O serviço de configuração é conduzido autonomamente de acordo com o tipo de configuração ativa no momento, e esta informação é bastante importante e faz muita diferença na sequência de passos a ser executada pelo serviço. A configuração estática prevê a ligação das VMs, verificação do estado do ambiente das mesmas e no fim da execução as VMs são desligadas. Já na configuração dinâmica, as VMs são criadas e configuradas com os dados inseridos pelo *tester*, o ambiente de cada uma é preparado em tempo real, e no fim estas são eliminadas da *cloud*.

Idealmente, quando as VMs se encontram desligadas, não consomem os recursos da *cloud* (apesar de serem guardados os dados associados à VM, como o *template* e os discos associados).

6 RESULTADOS

Para comprovar que a estrutura implementada tem impacto na *performance* do OpenNebula, é realizada uma monitorização dos valores do *Sunstone*. Também é possível verificar a *performance* de um computador da rede, controlando e comparando os valores do seu comportamento no momento em que os testes remotos do SIJ estão em execução, e no momento em que os testes não estão em execução e os agentes de testes estão desligados. Antes de prosseguir com os resultados, é realizada uma avaliação aos estados possíveis das VMs para a otimização dos recursos da *cloud*.

6.1 GESTÃO DOS ESTADOS DAS VMS

Quando o tipo de configuração de compilação é estático, é desejável que a VM seja desligada na *cloud*, mas é necessário entender o que significa desligar a VM, e quais são as opções que permitem tirar maior proveito da arquitetura implementada. As opções disponibilizadas pela API [38] encontram-se descritas na Tabela 6.

Ação	Descrição
shutdown	Ação de desligar a VM na *cloud*.
stopped	A VM está parada, o estado foi guardado e foi transferido, juntamente com a imagem do disco, para a *datastore* do sistema.
suspended	O mesmo que parado, mas os ficheiros ficaram no anfitrião para a opção de retomar a VM no mesmo.
poweroff	O mesmo que suspenso, mas não é gerado nenhum ficheiro de *checkpoint* e os ficheiros permanecem no anfitrião para inicialização da VM no mesmo.
undeployed	A VM está desligada, o disco é transferido para a *datastore* e a VM pode posteriormente retomada.

Tabela 6 - Opções de ação sobre VM

Das opções apresentadas anteriormente, foram testadas duas, a opção de *poweroff* e *undeployed*. Apenas foram testadas estas duas opções porque representavam os dois cenários: os ficheiros de configuração e o disco da VM ficarem no anfitrião ou serem guardados na *datastore* do OpenNebula. Os resultados da aplicação dos diferentes estados estão presentes nos próximos subcapítulos.

6.2 TEMPO DE CONFIGURAÇÃO DAS VMS

A configuração estática tem associada a si um total de catorze VMs que atuam como agentes de testes. A configuração dinâmica tem associada a si o total de VMs que o *tester* decidir. Para avaliar o tempo médio de configuração das VMs foi definido que a monitorização iria ocorrer com catorze VMs. Esta decisão implica criar mais catorze VMs, para estar de acordo com o propósito da configuração estática. Para não sobrecarregar a *cloud*, as VMs estáticas vão estar desligadas com o estado *undeployed*, de modo a ficarem excluídas da contagem nos anfitriões.

O tempo avaliado engloba todos os passos intermédios executados pelo serviço de configuração, desde a criação ou iniciação das VMs, configuração e verificação de serviços, execução de testes e eliminação das mesmas. Para simplificar o processo, apenas foram executados os testes de base de dados (848 testes). São comparados os tempos, representados na Tabela 7, relativamente aos cenários:

- Cenário com VMs estáticas e agentes de testes desligados em *poweroff*;
- Cenário com VMs estáticas e agentes de testes desligados em *undeployed*;
- Cenário com VMs dinâmicas e criação de agentes de testes.

Tipo	Tempo de execução
Cenário de VMs no estado poweroff	1 hora e 31 minutos (aprox.)
Cenário de VMs no estado undeployed	1 hora e 29 minutos (aprox.)
Cenário de criação e eliminação de VMs	1 hora e 52 minutos (aprox.)

Tabela 7 – Tempos de execução da configuração geral

Como pode ser observado na tabela anterior, o tempo é ligeiramente mais custoso relativamente à criação e eliminação de VMs, e os tempos de desligar as mesmas, quer ficando no estado *poweroff* como *undeployed,* são muito próximos. Os tempos detalhados por fase do ficheiro de configuração encontram-se no Anexo - Tempos de execução.

Para a avaliação da *performance* da *cloud* e de VMs da mesma, apenas foi considerada a configuração estática. Esta decisão deve-se ao facto de, no momento da configuração dinâmica, as VMs serem criadas e posteriormente eliminadas, ocupando temporariamente o mesmo espaço que as VMs de configuração estática ocupam quando o estado delas é *deployed/undeployed*. Isto é, os valores de memória e CPU (reais e alocados) dos anfitriões serão idênticos porque as VMs criadas têm as mesmas configurações (na *cloud* e nos próprios SO's) que as VMs estáticas.

6.3 PERFORMANCE NO SUNSTONE

Quando as VMs são desligadas, o primeiro impacto no módulo *web* (*Sunstone*) de gestão do OpenNebula está diretamente relacionado com o número de VMs ativas, pois este valor diminui. Para entender como são apresentados tais valores é necessário verificar os dados dos anfitriões do OpenNebula. Os dados dos anfitriões, relativos aos valores alocados por valores totais, encontram-se na Tabela 8.

OpenNebula	CPU Alocado	Memória Alocada	Estado
Anfitrião 1	1150 / 4800 (24%)	45GB / 251.7GB (18%)	ON
Anfitrião 2	0 / 0	0KB / -	OFF
Anfitrião 3	3050 / 2400 (127%)	68.9GB / 94.5GB (73%)	ON
Anfitrião 4	450 / 0	9GB / -	OFF
Anfitrião 5	5075 / 2400 (211%)	162.5GB / 189.1GB (86%)	ON

Tabela 8 – Anfitriões OpenNebula

No presente momento, a *cloud* tem 3 anfitriões ativos (de um total de 5). É possível verificar que, no **Anfitrião 1**, o valor de CPU alocado é 1150 e o valor de CPU máximo é 4800, representando uma taxa de alocação de 24%.

Já relativamente ao **Anfitrião 3**, a percentagem de CPU alocado é de 127%. A alocação de 127% é apenas teórica, porque os limites máximos são definidos pelos gestores da *cloud*. Uma alocação teórica com valor superior a 100% é designada de *overcommit* [39]. Neste cenário é permitida a alocação de mais recursos do que os que existem nos anfitriões físicos. Na prática isto significa que, caso todas as VMs ativas necessitem da capacidade total de recursos que lhe está atribuída, irá ocorrer a disputa pelos recursos existentes, e consequentemente a degradação da performance das mesmas. A utilização de *overcommit* em ambientes onde as VMs não exijam, de forma continuada, a total capacidade que lhes é alocada, permite que existam mais máquinas virtuais, sem que a performance das mesmas seja degradada de forma visível.

É necessário avaliar a implicação que o valor de VMs ativas tem na contagem final da soma dos valores reais e alocados de todos os anfitriões, em relação a CPU e memória.

Na página inicial do *Sunstone,* é possível ler os valores relativos aos anfitriões ativos, representados pela soma dos valores alocados e reais, divididos pelos valores máximos, tanto da CPU como da memória. De seguida serão abordados três cenários dos valores da carga dos anfitriões:

- Cenário 1: VMs estáticas e agentes de testes ligados, em execução de testes;
- Cenário 2: VMs estáticas e agentes de testes desligados em *poweroff*;
- Cenário 3: VMs estáticas e agentes de testes desligados em *undeployed*.

→ VMs ligadas e testes em execução

Na Figura 37 são apresentados os dados de CPU e memória, alocados e reais e as percentages dos mesmos.

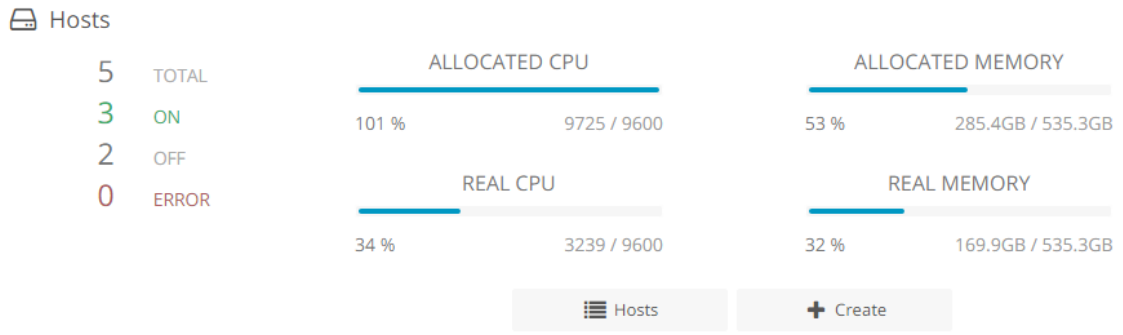

Figura 37 – Sunstone - Anfitriões com VMs ligadas

A percentagem de valores alocados equivale ao total alocado pelo total máximo, e a percentagem dos valores reais equivale ao total real pelo total máximo, tanto para os valores da CPU como da memória. A Figura 37 foi retirada num momento que as VMs definidas com uma configuração estática estavam ligadas na *cloud* e os testes remotos estavam em execução. Também é de salientar que esta figura foi retirada do *Sunstone* num dia útil, durante a tarde.

→ VMs desligadas e em estado *poweroff*

Num cenário em que as mesmas VMs estão desligadas, com o estado *poweroff*, então o cenário muda e os valores podem ser visualizados Figura 38.

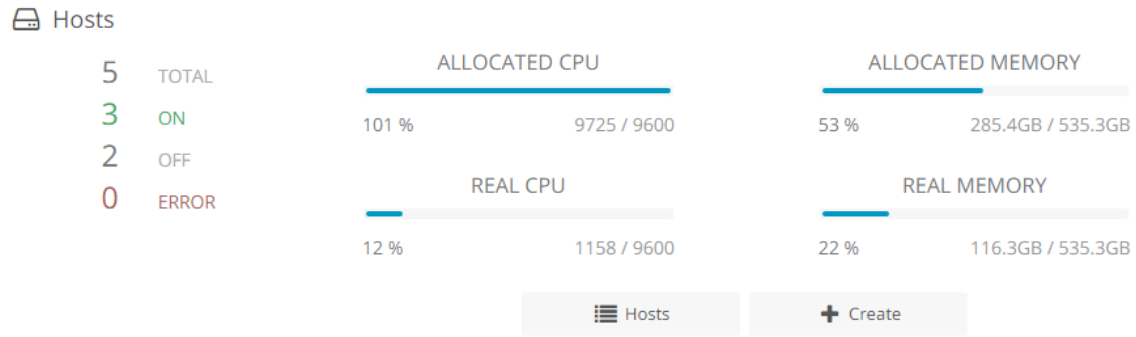

Figura 38 - Sunstone - Anfitriões com VMs desligadas (poweroff)

Os valores reais diminuem significativamente comparando com o cenário quando as VMs estão ligadas e em execução de testes (menos 22% no CPU e menos 10% na

memória) e os valores alocados mantêm-se. Isto significa que os recursos das VMs que foram desligadas foram libertados, porém continuam alocados em disco. Isto permite que as imagens sejam inicializadas em qualquer momento, no estado em que foram desligadas.

→ VMs desligadas e em estado *undeployed*

Já quando as VMs estão desligadas e os seus dados estão guardados na *datastore* e não nos anfitriões, os valores descem não só relativamente aos valores reais mas também aos valores alocados (Figura 39).

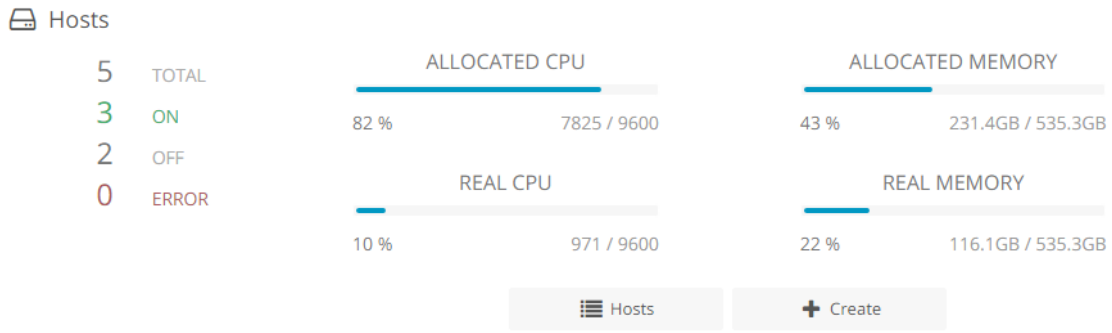

Figura 39 – Sunstone - Anfitriões com VMs desligadas (undeployed)

Comparando com os valores de quando as VMs estão ligadas e os testes estão em execução, a percentagem da CPU alocada desce 19%, e da CPU real desce 24%. A percentagem de memória alocada desde para 10% e de memória real desce também para 10%.Em comparação com o estado de *poweroff*, os valores descem (à exceção da memória real). Uma melhor representação destes valores encontra-se na Tabela 9.

A comparação dos valores brutos (e percentagens) dos cenários apresentados levam a uma perceção que a ação de colocar as VMs num estado *undeployed* é mais vantajoso do que colocar as VMs no estado *poweroff*. Para verificar essa suposição, é necessário analisar o comportamento e desempenho de VMs na própria estrutura da *cloud*, para verificar se há uma otimização dos recursos da mesma. A análise sobre o desempenho das VMs foi realizada durante os dias úteis de uma semana, pela tarde.

OpenNebula	VMs ligadas	VMs em *poweroff*	VMs em *undeployed*
CPU			
CPU alocado			
Total CPU Alocado	9725	9725	**7825**
Total CPU Máximo	9600	9600	9600
% CPU Alocado	**101%**	**101%**	**82%**
CPU real			
Total CPU Real	**3239**	**1158**	**917**
Total CPU Máximo	9600	9600	9600
% CPU Real	**34%**	**12%**	**10%**
Memória			
Memória alocada			
Total Memória Alocada	**285.4 GB**	**285.4 GB**	**231.4 GB**
Total Memória Máxima	535.3 GB	535.3 GB	535.3 GB
% CPU Alocado	**53%**	**53%**	**43%**
Memória real			
Total Memória Real	**162.1 GB**	**116.3 GB**	**116.1 GB**
Total Memória Máxima	535.3 GB	535.3 GB	535.3 GB
% Memória Real	**32 %**	**22%**	**22%**

Tabela 9 – Comparação de valores dos anfitriões

6.4 PERFORMANCE NO NOVABENCH

Para a monitorização das VMs do OpenNebula, foram efetuados testes de *benchmark*, utilizando o programa *NovaBench*, permitindo a avaliação de desempenho de algumas características de *software* e *hardware*. O *NovaBench* [40] permite testar os principais componentes da máquina virtual, através dos seguintes testes [41]:

- ***Floating Point Test*** - Testes de velocidade em operações aritméticas com vírgula flutuante;
- ***Integer Test*** **-** Testes de velocidade em operações aritméticas com inteiros;
- ***MD5 Hashing Speed*** **-** Testes gerais à CPU;
- ***3D Graphics Test*** - Testes à unidade de processamento gráfico;
- ***RAM Speed*** **-** Testes à velocidade de leitura e escrita na RAM;
- ***Disk Write Speed*** **-** Testes de velocidade de escrita no disco rígido primário.

A vista principal do programa está exposta na Figura 40.

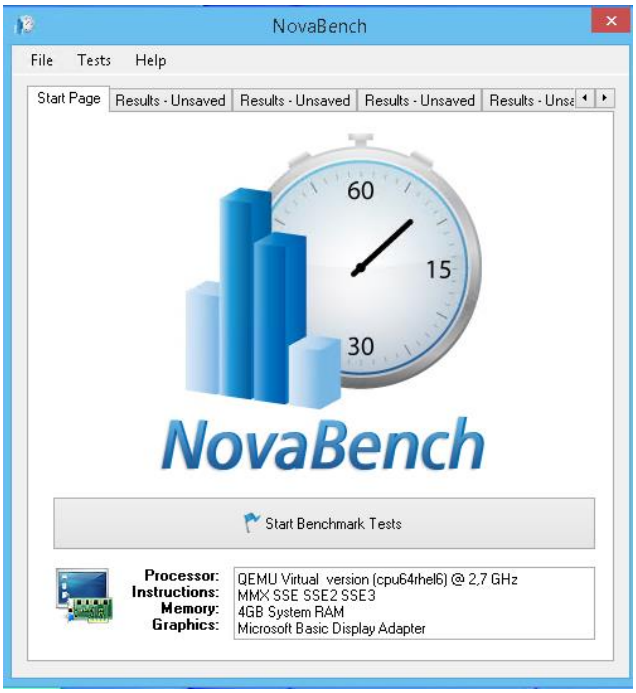

Figura 40 – NovaBench

Normalmente os testes demoram 1-2 minutos a executar e produzem como resultado informação detalhada dos testes e informação do sistema em geral. Este *software* já era

utilizado por gestores do OpenNebula e assim foi também utilizado na presente dissertação.

Estes testes realizados pelo NovaBench foram operacionalizados em três VMs diferentes. A **VM1** representa o servidor do domínio, tendo uma utilização muito frequente pelo *tester*. A **VM2** e **VM3** são máquinas utilizadas normalmente, cujo desempenho das mesmas é normalmente fraco. Na Tabela 10 encontram-se as características dos SO's das mesmas.

OpenNebula	Win. Version	Processador	RAM
VM1	Win. Server 2008 R2 Enterprise (64-bit)	QEMU Virtual CPU version (cpu64-rhel6) 2.67GHz	4 GB
VM2	Win 7 Enterprise (64-bit)	QEMU Virtual CPU version (cpu64-rhel6) 3.07GHz	2 GB
VM3	Win 8.1 Enterprise (64-bit)	QEMU Virtual CPU version (cpu64-rhel6) 2.67GHz	4 GB

Tabela 10 – Características VMs de benchmark.

Já na Tabela 11 é possível verificar as características das VMs, definidas pelo gestor da *cloud,* aquando da criação das mesmas.

OpenNebula	CPU	VCPU	Memória
VM1	4	4	4 GB
VM2	0.5	2	2 GB
VM3	2	2	4 GB

Tabela 11 – Características VMs (na cloud) de benchmark

Dos resultados que esta ferramenta disponibiliza, foram analisados os testes à velocidade da memória RAM e os testes de velocidade de escrita no disco primário.

Depois da monitorização e análise dos resultados recolhidos durante uma semana, sobre as três VMs em questão, nos três cenários, foi possível concluir que no cenário onde os agentes de testes estão desligados, no estado *undeployed*, os valores de desempenho melhoram consideravelmente. No Anexo - Performance no NovaBench é possível ver o registo destes dados, e nas duas imagens seguintes é possível ver um resumo dos resultados das leituras da velocidade da memória RAM e da escrita no disco, das três VMs (VM1, VM2 e VM3).

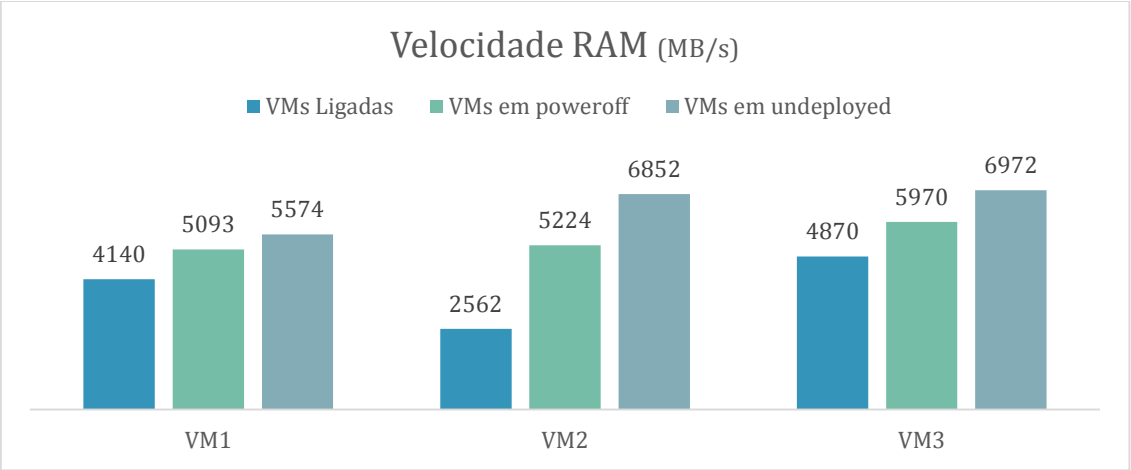

Figura 41 – Velocidade da memória RAM

De acordo com a Figura 41 é possível verificar que os valores são mais rápidos quando as VMs estão no estado *undeployed*, nas três VMs analisadas.

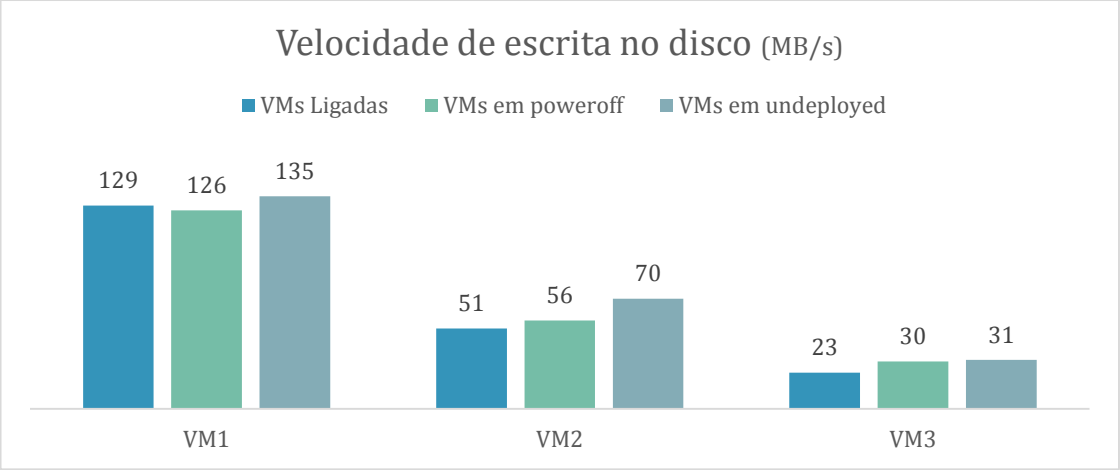

Figura 42 – Velocidade de escrita no disco

De acordo com a Figura 42 é possível verificar que, também para a velocidade de escrita no disco, os valores foram melhores nas VMs quando estão no estado *undeployed*.

6.5 CONSIDERAÇÕES FINAIS

Pelos resultados apresentados anteriormente é possível inferir que a otimização dos recursos na *cloud* tem um bom impacto no desempenho das VMs que estão na mesma. De acordo com as conclusões obtidas, foi decidido que as VMs permanecem desligadas durante o tempo em que os testes não necessitam de executar, e que as mesmas são inicializadas no momento de execução dos testes. Deste modo as VMs serão desligadas no estado *undeployed*, por melhorarem significativamente os valores reais e alocados de CPU e memória.

Em suma, entre os estados possíveis dos agentes de testes (ligadas *vs undeployed*), os valores alocados da CPU desceram 19% e valores reais da CPU desceram 24%. Já na memória, os valores alocados e reais desceram 10%.

Já os tempos de *performance* das três VMs monitorizadas aumentou, se compararmos os valores enquanto as VMs estão ligadas ou desligadas. No entanto é possível observar que os valores foram melhores quando as VMs estão desligadas no estado *undeployed* (Anexo - Performance no NovaBench).

7 CONCLUSÕES

Com a implementação da arquitetura proposta e após a análise dos resultados, chega o momento de verificar que problemas foram ultrapassados e que conclusões podem ser retiradas da aplicação da arquitetura apresentada nesta dissertação.

7.1 CONCLUSÕES

O objetivo deste estudo passava por desenhar e implementar uma arquitetura automatizada para integração de código, compilação e execução de testes de *software*. Os testes de *software*, que executam em máquinas virtuais da *cloud* (que tomam o papel de agentes de testes), permitem uma execução distribuída mais funcional. No âmbito desta dissertação foi proposto que, a cada execução de testes do SIJ, fosse possível gerir e otimizar os recurso da *cloud* onde estão alojados os agentes de testes.

Numa fase de desenvolvimento anterior, os testes do SIJ executavam a qualquer hora do dia, e os agentes de testes permaneciam sempre ligados na *cloud*, mesmo quando não estavam a ser necessários. De modo a aproveitar as vantagens da utilização de uma estrutura de *Cloud Computing*, é possível gerir os recursos, tornando-os elásticos e adaptáveis às necessidades de todos os utilizadores da *cloud*. Durante o dia, a *cloud* é ativamente utilizada para desenvolvimento e também para suporte a aulas da Universidade de Aveiro, e durante a noite, a carga da *cloud* diminui. É possível tirar vantagens da utilização diurna/noturna da *cloud*, para a execução de testes do SIJ, isto é, se durante o período noturno não há tanta necessidade de disponibilidade de recursos por parte dos utilizadores do OpenNebula, então os testes podem ser executados nesse mesmo horário, libertando os recursos quando os testes não estão em execução.

Perante este cenário, foi desenvolvida e implementada uma arquitetura que permite que os recursos sejam mais elásticos, para maior proveito, tanto da equipa de testes e como dos utilizadores normais da mesma *cloud*.

Os resultados da aplicação desta arquitetura foram divulgados no capítulo anterior.

Foi possível provar que a estrutura proposta provocou diferenças no comportamento de VMs gerais da *cloud*, quando os agentes de testes estão ligados e a execução dos testes do SIJ, e quando os mesmos estão desligados.

A equipa de testes decidiu utilizar o cenário no qual os agentes de testes ficam no estado *undeployed*, pois os resultados dos tempos de execução do ficheiro de configuração são melhores, e os resultados da *performance* de VMs gerais da *cloud* também.

7.2 PROBLEMAS ENCONTRADOS

Durante a realização deste estudo foram encontrados diferentes tipos de problemas, dos quais uns foram possíveis ultrapassar, e outros não.

Relativamente à gestão de questões do funcionamento interno dos agentes de testes, um dos maiores problemas acontece pela ocorrência muito frequente de falta de ligação de rede de domínio (Internet). Quando uma máquina virtual fica sem acesso à rede, só pode ser reparada a partir do VNC (manualmente). A implicação deste acontecimento para a estrutura apresentada é que a máquina virtual que atua como agente de teste, deixa de estar apta para a execução de testes. Se o controlador de testes fica sem acesso à rede, então a execução de testes não é inicializada e à equipa de testes é reportada a ocorrência para resolução manual.

No que concerne à gestão das máquinas virtuais que integrariam a estrutura como agentes de testes, também ocorreram alguns problemas de perda de ficheiros nos anfitriões e *datastores*. Mais concretamente, em algumas ações realizadas sobre as VMs, como *suspend*, *poweroff* ou *stop*, os ficheiros das VMs ficavam perdidos nos *datastores* da cloud. Nestes casos era necessário recorrer ao gestor do OpenNebula para que manualmente pudesse corrigir o problema, visto que esses problemas ficavam fora do âmbito de resolução por parte da equipa de testes. Este problema pode acontecer devido à sequência de passos realizados entre a ação pretendida, desde a API até ao *hypervisor*. As ações invocadas a partir da API XML-RPC.NET, de Charles Cook, são comunicadas para o OpenNebula. Esta comunicação é traduzida, pelo OpenNebula,

para a forma de um comando que será executado pelo *hypervisor*. A tradução (ou traduções) realizada pode levar a erros de interpretação ou perda de dados em qualquer um dos interpretadores por qual o processo passa. Normalmente a resolução destes problemas implica a análise do ficheiro de *logs*, que fica registado na *cloud*.

No que diz respeito ao TFS e controlador de compilação, foi encontrado um problema com a *Framework* .NET da máquina virtual responsável pelo projeto do SIJ, no TFS, isto é, a máquina virtual que atua como controlador da compilação do VS. O problema está relacionado com a versão da *Framework* .NET pois estava desatualizada. Como solução a este problema, foi requerido ao gestor dessa mesma máquina que fizesse a sua atualização.

O agendamento de uma compilação da solução, a partir do VS, ainda não é possível, devido a problemas relacionados com a localização das *assemblies* instaladas no controlador da compilação. Na prática, este erro está relacionado com o VS não conseguir encontrar, na VM do controlador da compilação, as *assemblies* necessárias para compilação dos projetos da solução do SIJ. Deste modo, e até ser possível solucionar o problema, o serviço de configuração é executado na compilação manual, realizada por um membro autorizado da equipa. Para executar os testes remotos, basta ao programador adicionar a localização do ficheiro *.bat*, no evento de pós compilação da solução do VS.

7.3 TRABALHO FUTURO

Como trabalho futuro é sugerido que sejam resolvidos os problemas de dependências, na VM onde se encontra registado o controlador de compilação, para que seja possível estabelecer a comunicação entre o mesmo e o VS. Deste modo, será possível agendar a compilação automática no VS, sem existir a necessidade de esta ser executada manualmente.

7.4 CONSIDERAÇÕES FINAIS

Durante o desenvolvimento desta dissertação foi escrito um artigo científico, relacionado com a prática de *Continuous Integration* e a utilização da *Cloud Computing* para execução de testes de *software*, intitulado de "*Continuous Integration using Cloud Computing*" [1]. Este artigo foi aceite e apresentado na 10ª Conferência Ibérica em Tecnologias e Sistema de Informação, realizada este ano em Águeda.

Depois de toda a análise e conclusões é possível perceber que o melhor cenário para a otimização dos recursos da *cloud* e para a elasticidade dos mesmos acontece quando os agentes de testes são ligados e desligados, e quando a ação de desligar toma a ação de *undeployed* e permite que os recursos dos anfitriões sejam libertados.

8 BIBLIOGRAFIA

[1] J. C. Vigário, C. Teixeira e J. Sousa Pinto, "Continuous integration using cloud computing," em *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference*, Aveiro, 2015.

[2] J. A. A. Brazeta, "Testes de Software no Sistema de Informação da Justiça Cabo-Verdiana," em *Testes de Software no Sistema de Informação da Justiça Cabo-Verdiana*, 2014, pp. 2-3;9-10;44-45;57.

[3] C. Sandler, T. Badgett e G. J. Myers, The Art of Software Testing, New Jersey: JohnWiley & Sons, Inc., 2012.

[4] S. Buddies, "The Engineering Design Process," Science Buddies, [Online]. Available: http://www.sciencebuddies.org/engineering-design-process/engineering-design-process-steps.shtml. [Acedido em 2014 novembro 12].

[5] Microsoft, "Team Foundation Server," Microsoft, 2013. [Online]. Available: https://msdn.microsoft.com/en-us/vstudio/ff637362.aspx. [Acedido em 16 setembro 2014].

[6] J. Rosa, C. Teixeira e J. Sousa Pinto, "Risk factors in e-justice information systems," *Gov. Inf. Q.,* vol. 30, nº 3, pp. 241-256, julho 2013.

[7] J. Sousa Pinto e C. Teixeira, "Assisted On-Job Training," em *E-Learning - Engineering, On-Job Training and Interactive Teaching*, Croatia, Intech, 2012, p. 23.

[8] Microsoft, "Microsoft SQL Server.," Microsoft, 2014. [Online]. Available: https://msdn.microsoft.com/en-us/library/bb545450.aspx. [Acedido em outubro 2014].

[9] Microsoft, "Entity Framework (EF) Documentation," Microsoft, 2015. [Online]. Available: https://entityframework.codeplex.com/. [Acedido em setembro 2015].

[10] T. Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall PTR, 2005.

[11] Microsoft, “What Is Windows Communication Foundation,” Microsft, 2014. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx. [Acedido em outubro 2014].

[12] J. Pan, “Software Testing,” Carnegie Mellon University - Dependable Embedded Systems, 1999. [Online]. Available: http://users.ece.cmu.edu/~koopman/des_s99/sw_testing/. [Acedido em 15 setembro 2015].

[13] Microsoft, “Verifying Code by Using UI Automation,” Microsoft, 2014. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd286726(v=vs.120).aspx. [Acedido em outubro 2014].

[14] Microsoft, “Active Directory Domain Services,” Microsoft, 9 agosto 2009. [Online]. Available: https://technet.microsoft.com/en-us/library/cc753910.aspx. [Acedido em 19 setembro 2015].

[15] P. Duvall, S. Matyas e A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.

[16] M. Kawalerowicz e C. Berntson, Continuous Integration in .NET, Stamford: Manning, 2011.

[17] M. Fowler, “Continuous Integration,” p. 1, 01 maio 2006.

[18] E. Blankenship, M. Woodward, G. Holliday e B. Keller, “Professional Team Foundation Server 2012,” em *Professional Team Foundation Server 2012*, Indianapolis, John Wiley & Sons, Inc, 2013, pp. 3-4; 29-31; 495-500.

[19] M. Mason, em *Pragmatic Guide to Subversion*, Pragmatic Programmers, LLC, 2010, pp. 9-12; 18-19.

[20] T. Swicegood, “Introduction,” em *Pragmatic Guide to Git*, Pragmatic Programmers, LLC., 2010, pp. 9-16.

[21] “TortoiseGit - The coolest Interface to Git Version Control,” TortoiseGit, 2013. [Online]. Available: https://code.google.com/p/tortoisegit/. [Acedido em 19 janeiro 2015].

[22] J. Ehn e T. Sandstrøm, "So, what is Team Foundation Server 2012?," em *Team Foundation Server 2012 Stater*, Birmingham, Packt Publishing, 2012, pp. 1-2.

[23] "About CruiseControl.NET (abbr. CCNet)," CruiseControl.NET, 2011. [Online]. Available: http://www.cruisecontrolnet.org/projects/ccnet/wiki/About. [Acedido em 19 janeiro 2015].

[24] J. F. Smart, "Introducing Jenkins," em *Jenkins:The Definitive Guide*, Sebastopol,, O'Reilly Media, 2011, pp. 1-3.

[25] A. M. Berg, "Preface," em *Jenkins Continuous Integration Cookbook*, Birmingham, Packt Publishing, 2012, pp. 1-2; .

[26] E. Blankenship, M. Woodward, G. Holliday e B. Keller, "Overview of Build Automation," em *Professional Team Foundation Server 2012*, Indianapolis, John Wiley & Sons, Inc, 2013, pp. 377-380.

[27] J. Rossberg e M. Olausson, Pro Application Lifecycle Management with Visual Studio 2012, Apress, 2012.

[28] B. Soninsky, Cloud Computing Bible, Indianapolis: Wiley Publishing, Inc, 2011.

[29] C. Popoviciu, "How do cloud elasticity and cloud scalability differ," TechTarget, fevereiro 2013. [Online]. Available: http://searchtelecom.techtarget.com/answer/How-do-cloud-elasticity-and-cloud-scalability-differ. [Acedido em setembro 2015].

[30] O. Project, "An Overview of OpenNebula," OpenNebula, 2015. [Online]. Available: http://docs.opennebula.org/4.8/design_and_installation/building_your_cloud/intro.html. [Acedido em 05 janeiro 2015].

[31] D. M. Aranda, "An Introduction to Cloud Computing with OpenNebula," 19 novembro 2013. [Online]. Available: http://www.slideshare.net/opennebula/tecni-28445050. [Acedido em 10 fevereiro 2015].

[32] O. Project, "OpenNebula Sunstone: The Cloud Operations Center," OpenNebula Project, 2015. [Online]. Available:

http://docs.opennebula.org/4.12/administration/sunstone_gui/sunstone.html. [Acedido em 15 novembro 2014].

[33] O. Project, "The Filesystem Datastore," OpenNebula Project, 2015. [Online]. Available: http://docs.opennebula.org/4.12/administration/storage/fs_ds.html. [Acedido em janeiro 2015].

[34] O. Project, "KVM Driver," OpenNebula Project, 2015. [Online]. Available: http://docs.opennebula.org/4.12/administration/virtualization/kvmg.html. [Acedido em fevereiro 2015].

[35] O. Project, "Integration," OpenNebula, 2015. [Online]. Available: http://docs.opennebula.org/4.8/integration/getting_started/introapis.html. [Acedido em 23 janeiro 2015].

[36] O. Project, em *OpenNebula 4.10 Design and Installation Guide, Release 4.10.2*, 2015, p. 1;5;26.

[37] C. Cook, "XML-RPC.NET," 17 abril 2011. [Online]. Available: http://xml-rpc.net/. [Acedido em 10 fevereiro 2015].

[38] O. Project, "Managing Virtual Machines," OpenNebula Project, 2015. [Online]. Available: http://docs.opennebula.org/4.12/user/virtual_resource_management/vm_guide_2.html. [Acedido em 10 março 2015].

[39] T. Target, "memory overcommit (or overcommitment) definition," Tech Target, Outubro 2009. [Online]. Available: http://searchservervirtualization.techtarget.com/definition/memory-overcommit. [Acedido em 21 Dezembro 2015].

[40] N. Inc, "About NovaBench," Novawave Inc, 2015. [Online]. Available: https://novabench.com/about.php. [Acedido em 03 maio 2015].

[41] Pplware, "NovaBench 3.0 – Teste o desempenho geral da sua máquina," Pplware, 29 maio 2010. [Online]. Available: http://pplware.sapo.pt/software/novabench-3-0-teste-o-desempenho-geral-da-sua-maquina/. [Acedido em setembro 2015].

[42] "NIST General Information," National Institute of Standards and Technology (NIST), 12 maio 2015. [Online]. Available: http://www.nist.gov/public_affairs/general_information.cfm. [Acedido em 22 setembro 2015].

[43] Microsoft, "Create a team project," Microsoft, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms181477(v=vs.120).aspx. [Acedido em 26 julho 2015].

[44] Microsoft, "How Windows PowerShell Works," Microsoft, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms714658(VS.85).aspx. [Acedido em 20 fevereiro 2015].

[45] Microsoft, "Manage team project collections," Microsoft, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd236915(v=vs.120).aspx. [Acedido em 26 julho 2015].

[46] Microsoft, "Test Results Reported," Microsoft, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms182495(v=vs.100).aspx. [Acedido em abril 2015].

[47] Microsoft, "Using NSlookup.exe," Microsoft, 2015. [Online]. Available: https://support.microsoft.com/en-us/kb/200525. [Acedido em 1 outubro 2015].

[48] M. TechNet, "Ping," Microsoft TechNet, 2015. [Online]. Available: https://technet.microsoft.com/en-us/library/cc940091.aspx. [Acedido em 1 outubro 2015].

[49] W3Schools, "ASP.NET MVC Tutorial," W3Schools, 2015. [Online]. Available: http://www.w3schools.com/aspnet/mvc_intro.asp. [Acedido em outubro 2015].

[50] W3Schools, "Bootstrap Get Started," W3Schools, 2015. [Online]. Available: http://www.w3schools.com/bootstrap/bootstrap_get_started.asp. [Acedido em outubro 2015].

[51] W3Schools, "XML Soap," W3Schools, 2015. [Online]. Available: http://www.w3schools.com/xml/xml_soap.asp. [Acedido em 1 October 2015].

9 ANEXOS

9.1 DIAGRAMA DE ARQUITETURA DE CI PARA O SIJ

Na seguinte imagem é possível verificar o diagrama de arquitetura completo.

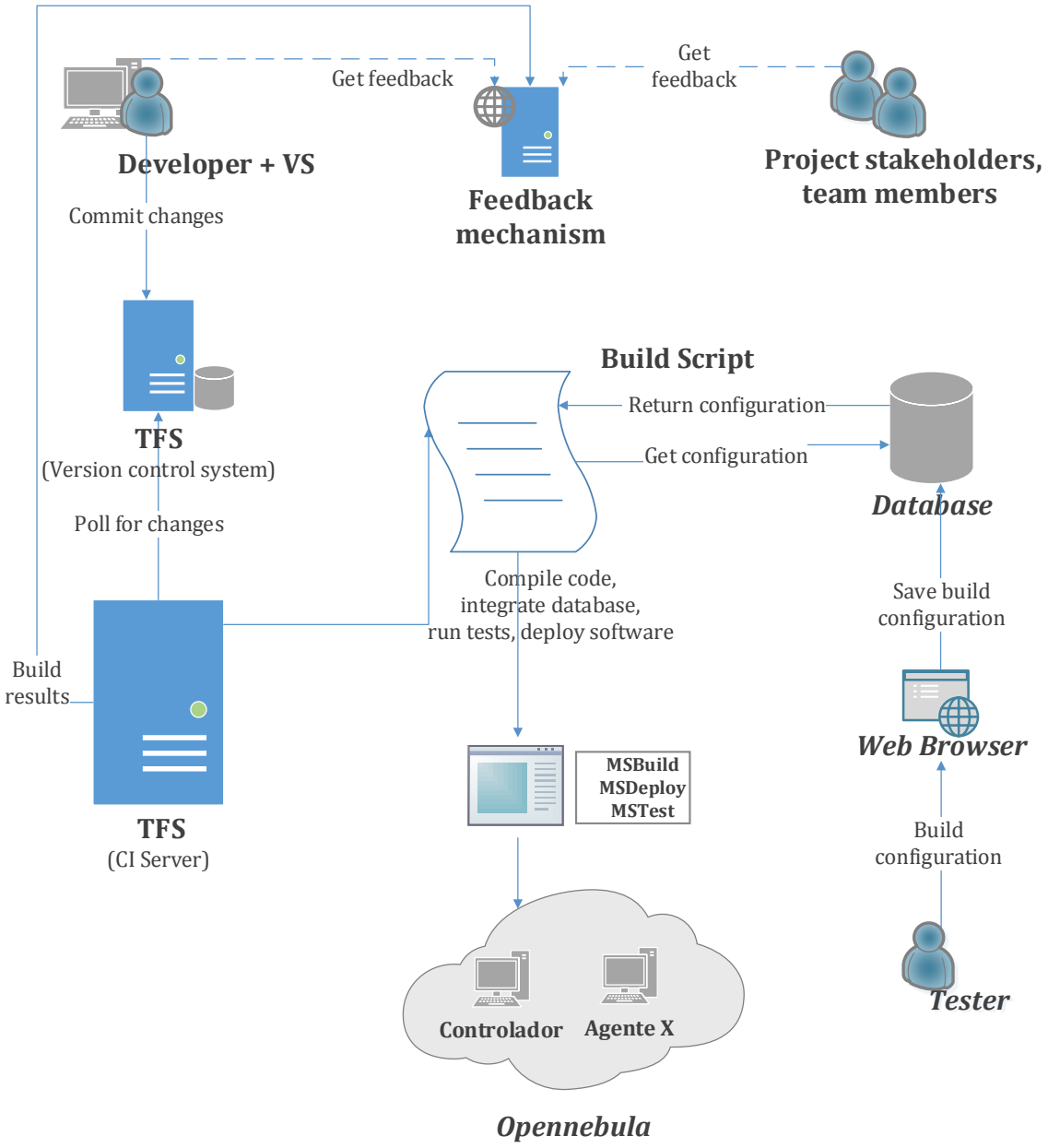

Figura 43 – Diagrama de arquitetura de CI para o SIJ

Todas as componentes foram descritas nos capítulos anteriores. Em suma, o TFS comporta-se como servidor de CI e sistema de controlo de versões. É possível a implementação de uma estrutura de CI, que comunica com os agentes de testes do OpenNebula a partir de um serviço de configuração que faz toda a gestão do processo mais complexo. As informações que são consumidas pelo ficheiro de configuração são definidas pelo *tester*, a partir de uma aplicação web, desenvolvida unicamente para suporte da compilação automática.

9.2 MODELO DE DADOS

A Figura 44 representa toda a estrutura de modelo de dados, a qual serve de apoio para a execução do serviço de configuração. De seguida são enumeradas todas as tabelas, acompanhadas de uma pequena descrição, informando do papel de cada uma.

◊ ***AuthorizedUser*:** Detalhes do utilizador autorizado a executar a compilação automática;

◊ ***Build*:** Detalhes da compilação automática, como se está ativa ou as datas de início e fim. Associada a si, tem os resultados dos testes, o registo geral das VMs utilizadas para a compilação e os dados relativos aos serviços de SQL de cada VM;

◊ ***BuildConfig*:** Detalhes da configuração da compilação, como se está ativa ou quantas VMs vão executar os testes. Associados a cada configuração existe um tipo de configuração, uma informação genérica e um utilizador autorizado;

◊ ***BuildConfigType*:** Tipo de configuração da compilação, que tem associados a si uma configuração de compilação e um conjunto de VMs;

◊ ***CPUDetails*:** Detalhes associados à CPU dos anfitriões do OpenNebula, onde os registos estão associados à *performance* e são gravados no início e fim de cada compilação;

◊ ***GenericInfo*****:** Detalhes gerais da configuração, relacionados com o SIJ e com o OpenNebula;

◊ ***MEMDetails***: Detalhes associados à memória dos anfitriões do OpenNebula, onde os registos estão associados à *performance* e são gravados no início e fim de cada compilação;

◊ ***Performance*****:** Detalhes associados à performance dos anfitriões, registados no início e fim de cada compilação e permitem ver detalhes dos dados da CPU e memória;

◊ ***TestsResults*****:** Dados relacionados com os resultados de cada tipo de teste executado, como número de testes executados, que passaram ou falharam;

◊ ***TestType*****:** Tipo de testes relacionados com a execução de testes, onde estão associados os resultados dos testes;

◊ ***VM*****:** Detalhes de cada VM associada à compilação automática, onde são registados dados relativos ao OpenNebula;

◊ ***VMLog*****:** Informação geral dos estados das VMs utilizadas para a compilação automática;

◊ ***VMSet:*** Conjunto de VMs definidas pelo gestor da compilação e associadas ao tipo de configuração.

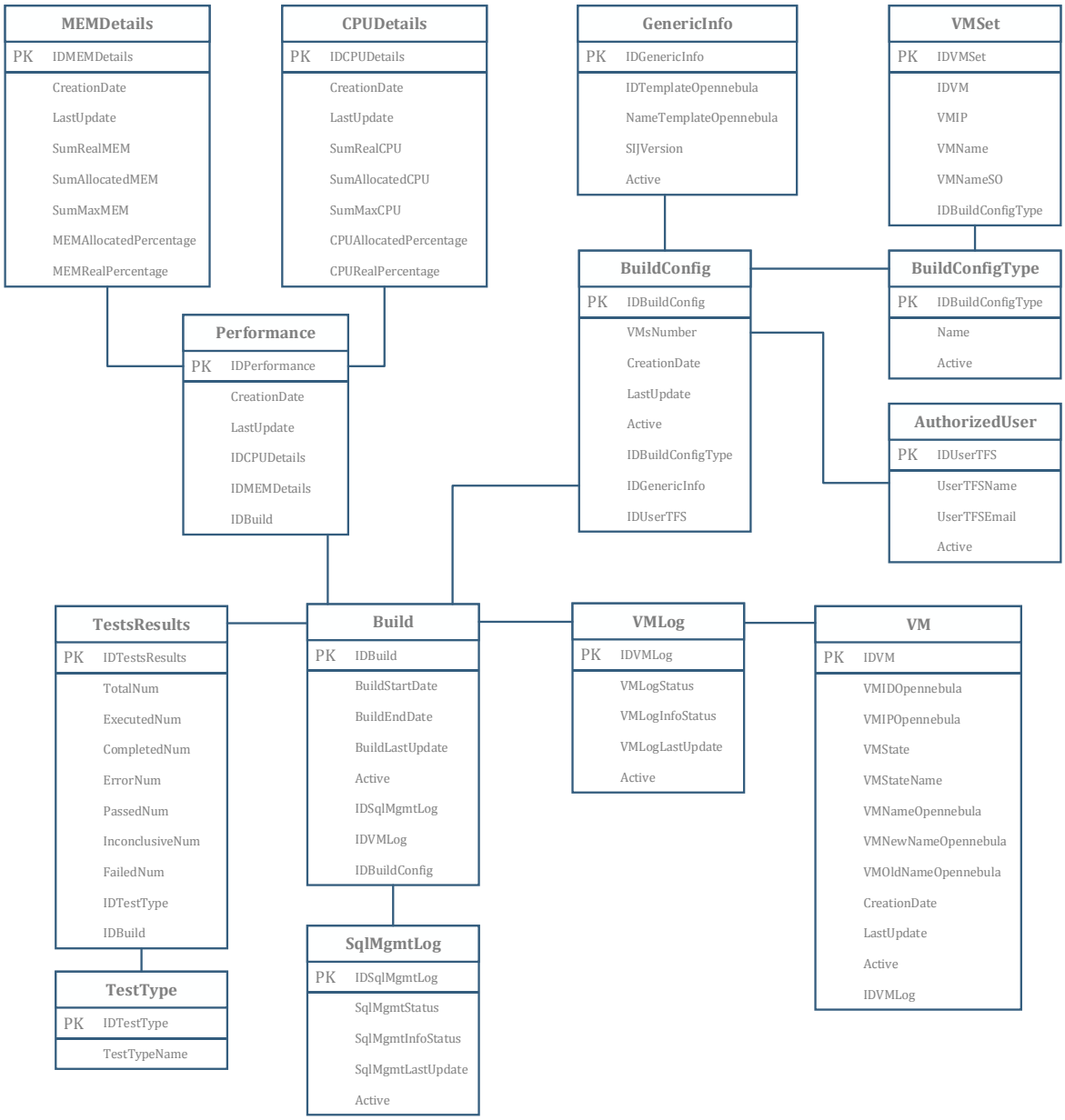

Figura 44 – Modelo de Dados

9.3 FLUXO DE COMPILAÇÃO AUTOMÁTICA

No fluxo apresentado na Figura 45 é exposto o agendamento e execução de uma compilação automática. A interação é realizada com o VS, onde é assinalada uma compilação automática, e é estabelecida a ligação com o controlador de compilação.

Se a ligação com o controlador tiver sucesso, então o processo de compilação dá início, executando um evento de pós-compilação, que invoca o serviço de configuração da compilação. Este serviço está responsável pela gestão de recursos e execução dos testes de *software*.

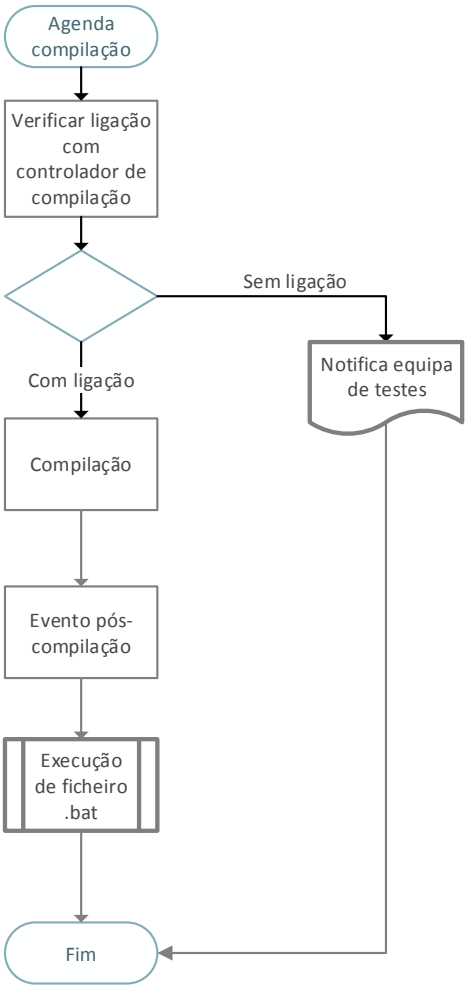

Figura 45 – Fluxo de compilação

9.4 DIAGRAMAS DE SEQUÊNCIA

Os diagramas de sequência pretendem representar a prossecução de passos no serviço de configuração. Nos próximos subcapítulos são divulgados os mesmos diagramas que foram explicados no subcapítulo Estruturação do serviço de configuração.

9.4.1 Diagrama sequência: Módulo de gestão do OpenNebula

O diagrama de sequência representado na Figura 46 tem a intenção de explicar como é definido o comportamento do ficheiro relativamente à configuração da compilação.

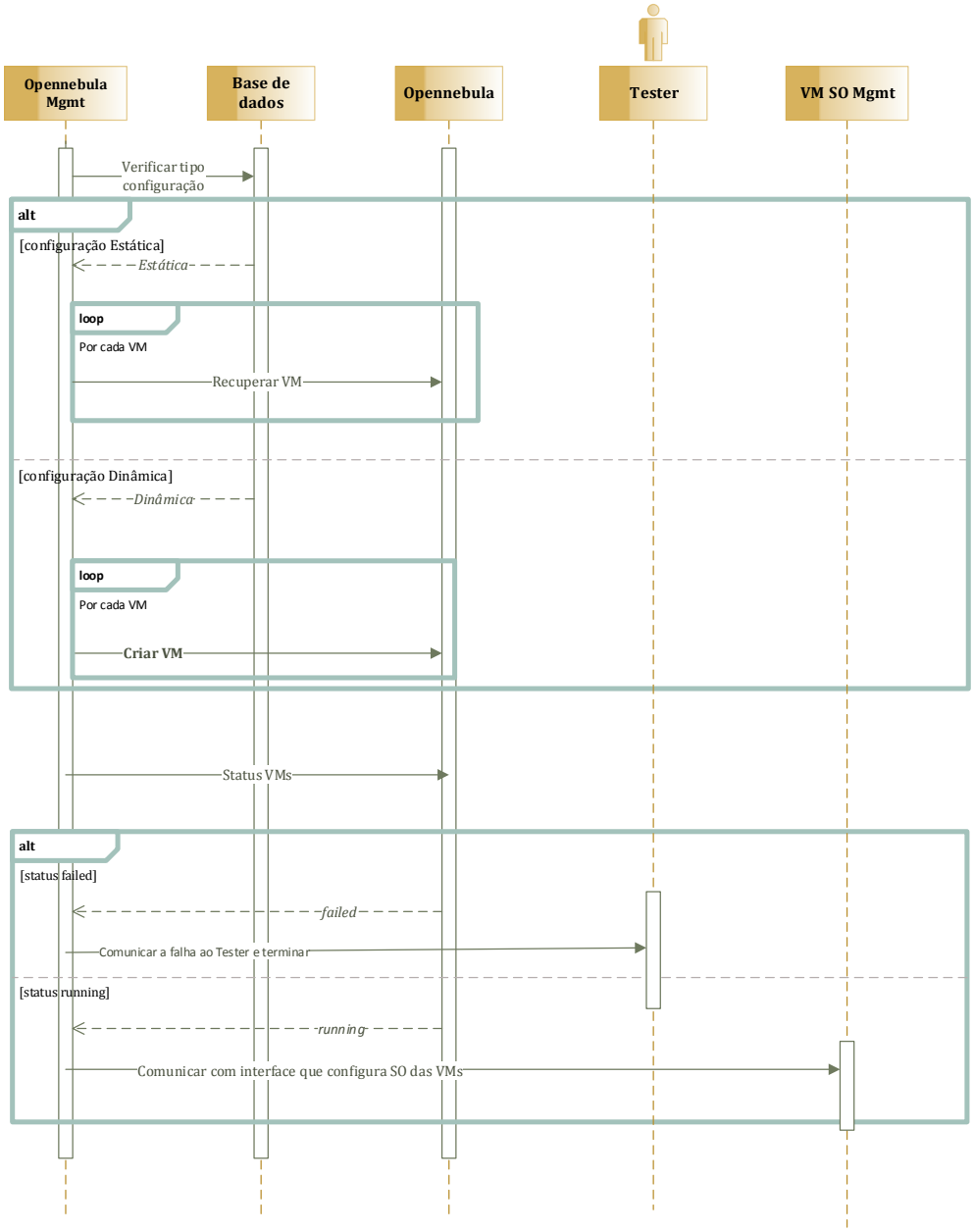

Figura 46 – Diagrama sequência: módulo de gestão do OpenNebula

Se a configuração for dinâmica, este módulo é responsável por criar e publicar as VMs na *cloud*. Já se a configuração for estática, então as VMs têm de ser recuperadas e tem de existir uma verificação do estado (correto) das VMs.

9.4.2 Diagrama sequência: Módulo de gestão de SO das VMs

Este diagrama (Figura 47) está responsável pela configuração dos SOs das VMs. Quando a configuração é dinâmica, o comportamento deste módulo é essencial porque é ele que insere as novas VMs no domínio do OpenNebula.

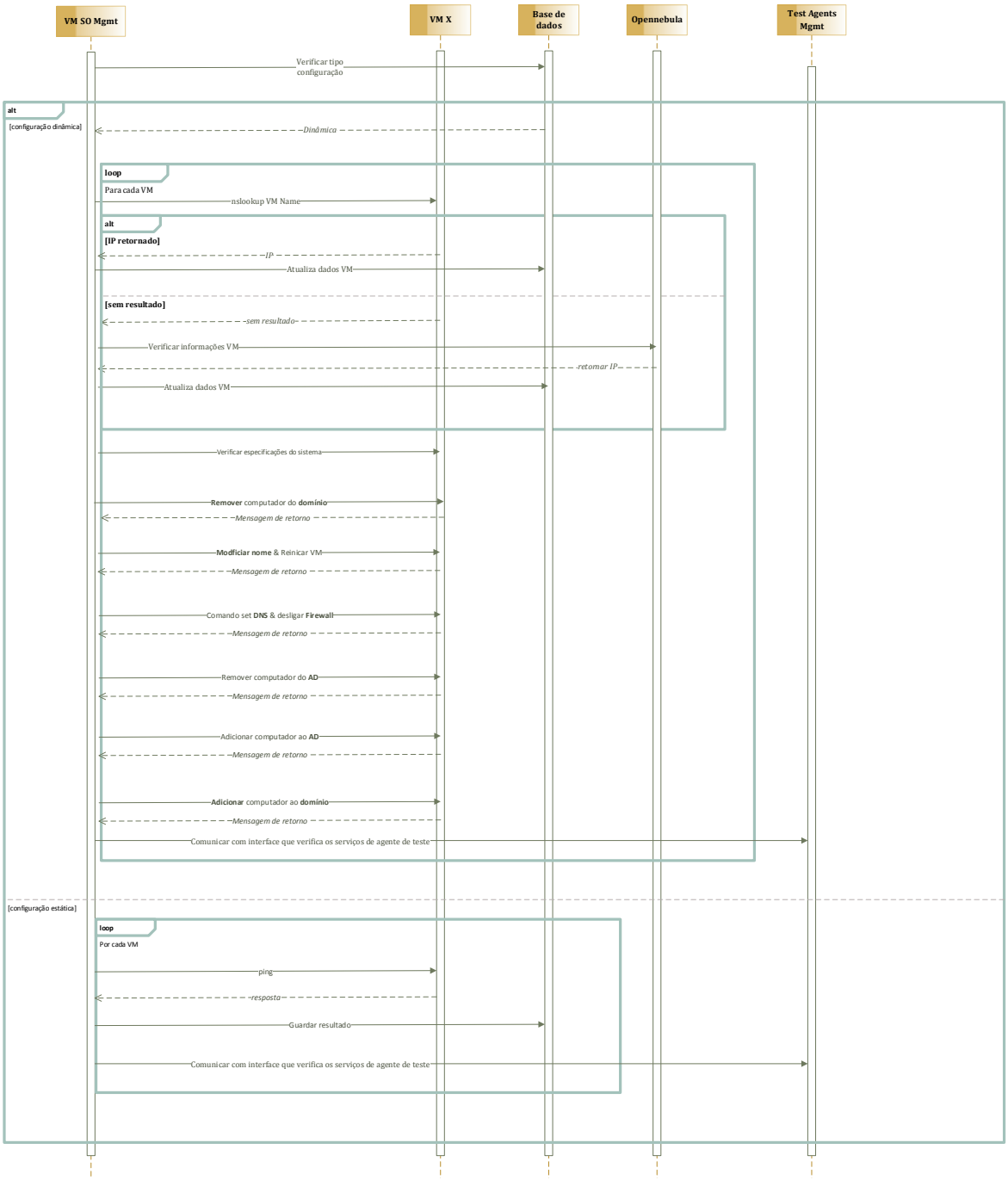

Figura 47 – Diagrama sequência: módulo de gestão do SO das VMs

Deste modo é possível que os testes de *software* sejam distribuídos e executados pelos novos agentes de testes.

9.4.3 Diagrama sequência: Módulo de gestão de processos de agente de testes

O módulo de gestão dos processos de agentes de testes garante que os agentes estão ligados e aptos para a execução dos testes remotos.

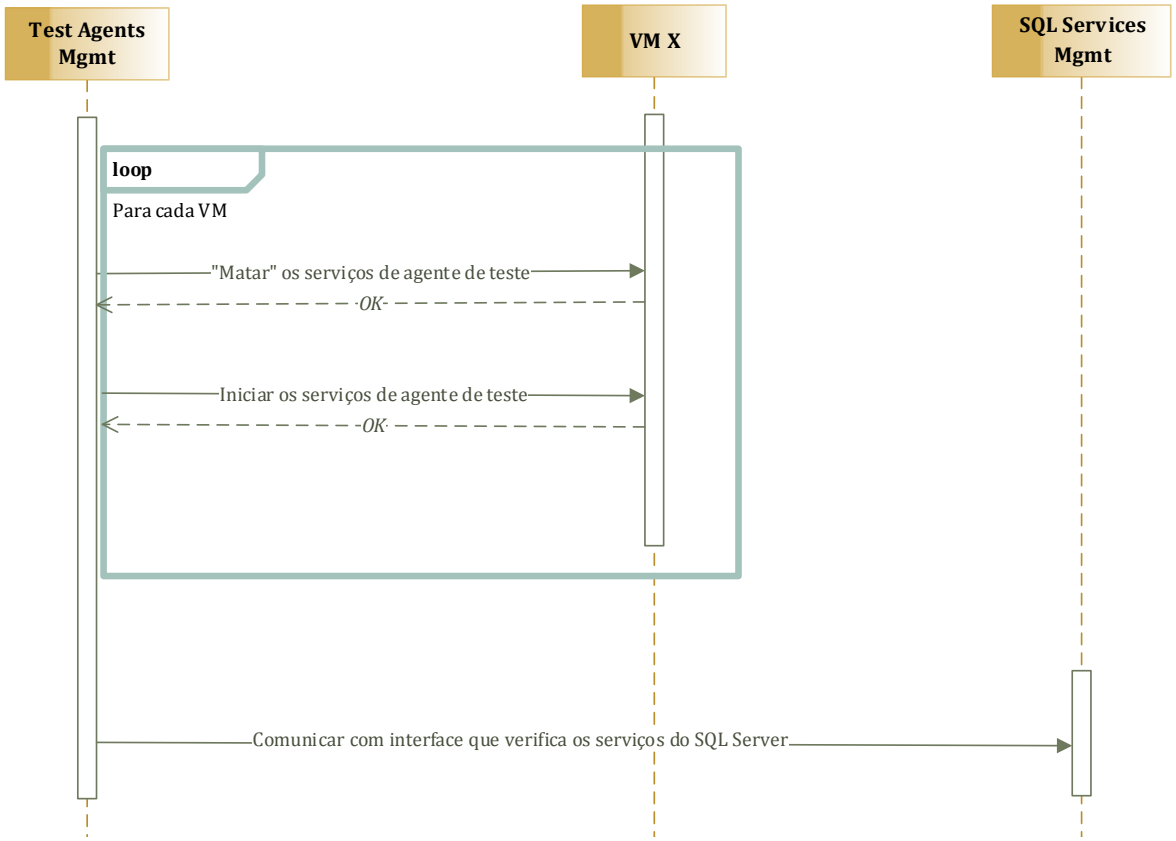

Figura 48 – Diagrama sequência: módulo de gestão de processos de agentes de testes

9.4.4 Diagrama sequência: Módulo de gestão de serviços SQL

O módulo da gestão de serviços SQL permite que os mesmos estejam a executar em cada uma das VMs, para ser possível comunicação entre as bases de dados e os testes que as utilizam. A perda da ligação destes serviços é frequente quando os agentes de

testes são desligados, e assim sendo, este módulo é essencial para o correto funcionamento dos testes de *software*.

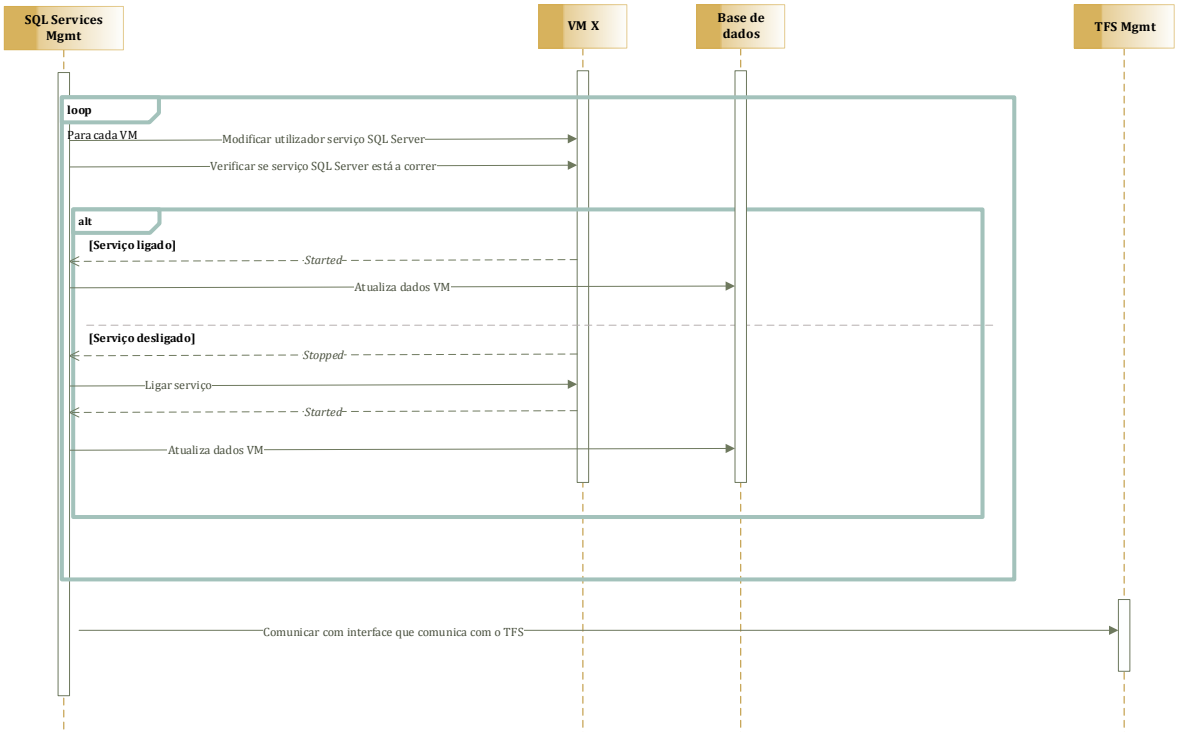

Figura 49 – Diagrama sequência: módulo de gestão de serviços SQL

9.4.5 Diagrama sequência: Módulo de gestão do TFS e execução de testes

Este módulo permite a realização da execução dos testes remotos. Antes dessa execução, ele garante que os ambientes dos agentes de testes estão aptos para a execução dos mesmos. Isto é, garante que em todas as VMs, a versão da base de dados e a versão de ficheiros de interface é a mais atual e os serviços que suportam o funcionamento do sistema (serviço *MJCVService*, *Queue* e *MJCVWorkflow*) estão atualizados e em execução. Esta garantia é conseguida a partir da execução de ficheiros Powershell que comunicam com os agentes de testes. Após o término da execução destes ficheiros, há uma verificação à base de dados para conhecer que tipos de testes vão ser executados. Quando a execução dos testes termina, os resultados são guardados e é realizada a análise do ficheiro dos resultados dos testes, que permite a extração dos valores dos resultados, como quantos testes foram executados, com sucesso e com insucesso, entre outros.

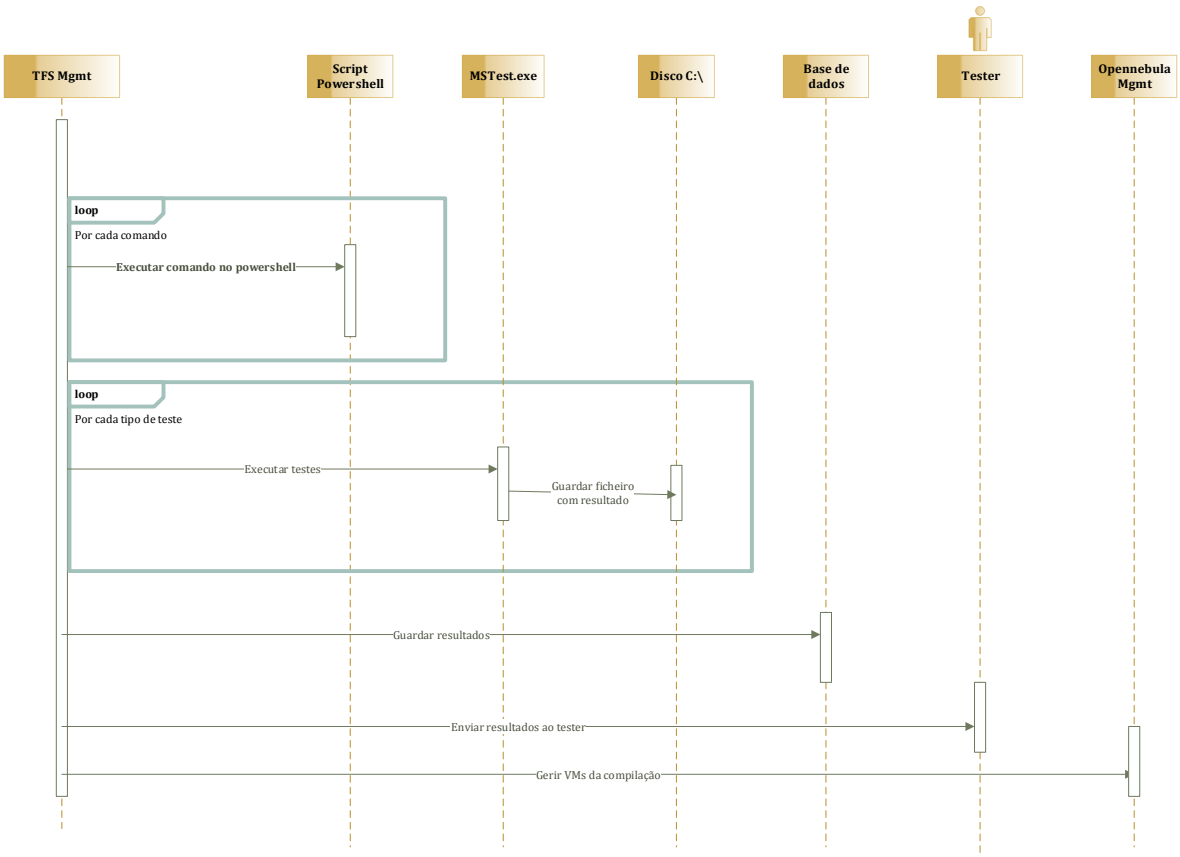

Figura 50 – Diagrama sequência: módulo de gestão do TFS e execução de testes

9.5 RESULTADOS

Por fim, foi necessário obter e analisar os resultados da arquitetura implementada. Nos próximos subcapítulos são divulgados os dados detalhados dos tempos de execução do serviço de configuração. São apresentados os dados para os três cenários:

- Cenário 1: VMs estáticas e agentes de testes desligados em *poweroff*;
- Cenário 2: VMs estáticas e agentes de testes desligados em *undeployed*;
- Cenário 3: VMs dinâmicas e criação de agentes de testes.

9.5.1 Tempos de execução do ficheiro de compilação

A Tabela 12 é referente aos dados vigiados para a configuração estática, com os agentes de testes previamente desligados pela ação de *poweroff*.

Tipo	Tempo de execução
Interface de configuração de compilação	27 segundos
Interface de gestão do OpenNebula	43 segundos
Interface de gestão de SO's	210 segundos
Interface de gestão de agente de testes	10 segundos
Interface de gestão de serviços SQL	30 segundos
Módulo de gestão do TFS	4524 segundos (aprox. 1 hora e 15 min.)
Término do processo	22 segundos
	1 hora e 21 minutos (aprox.)

Tabela 12 – Resultados do tempo de execução do cenário 1

A Tabela 13 é referente aos dados vigiados para a configuração estática, com os agentes de testes previamente desligados pela ação de *undeployed*.

Tipo	Tempo de execução
Módulo de configuração de compilação	26 segundos
Módulo de gestão do OpenNebula	75 segundos
Módulo de gestão de SO's	80 segundos
Módulo de gestão de agente de testes	7 segundos
Módulo de gestão de serviços SQL	20 segundos
Módulo de gestão do TFS	4524 segundos (aprox. 1 hora e 15 min.)
Término do processo	30 segundos
	1 hora e 19 minutos (aprox.)

Tabela 13 – Resultados do tempo de execução do cenário 2

A Tabela 14 é referente aos dados conseguidos para a configuração dinâmica, quando as VMs não existiam, e foram criadas no início da compilação automática, configuradas em todo as fases necessárias e eliminadas no final da execução.

Tipo	Tempo de execução
Módulo de configuração de compilação	25 segundos
Módulo de gestão do OpenNebula	120 segundos
Módulo de gestão de SO's	1644 segundos
Módulo de gestão de agente de testes	10 segundos
Módulo de gestão de serviços SQL	60 segundos
Módulo de gestão do TFS	4874 segundos (aprox. 1 hora e 35 min.)
Término do processo	40 segundos
	1 hora e 53 minutos (aprox.)

Tabela 14 – Resultados do tempo de execução do cenário 3

Os resultados dos tempos em que é necessário recuperar os agentes de testes são muito melhores que o tempo de criação e eliminação das VMs.

Nessa perspetiva, a equipa de testes optou por adotar o cenário 2, quando as VMs são desligadas e coladas em estado *undeployed*.

9.5.2 Performance no NovaBench

Os dados apresentados nas próximas tabelas foram recolhidos durante uma semana. Assim, as VM1, VM2 e VM3 foram monitorizadas recorrendo à utilização da ferramenta NovaBench, e os cenários destes testes dividem-se em:

- Cenário 1: VMs estáticas e agentes de testes ligados, em execução de testes;
- Cenário 2: VMs estáticas e agentes de testes desligados em *poweroff*;
- Cenário 3: VMs estáticas e agentes de testes desligados em *undeployed*.

Os resultados foram positivos e superiores quando os agentes de testes estavam desligados pela ação de *undeployed*. É apresentada a relação (em percentagem) do valor do estado *poweroff* e *undeployed* com o valor quando os agentes de testes estavam ligados e a executar os testes.

Os resultados da VM1 estão representados na seguinte tabela. Os resultados são melhores quando os agentes de testes estavam desligados para ação de *undeployed*.

	VMs ligadas c/ testes	VMs em *poweroff*	VMs em *undeployed*
4096 MB System RAM			
Velocidade RAM	4140 MB/s	5093 MB/s	5574 MB/s
Relação com VMs ligadas		23%	35%
Hardware Tests			
Velocidade de escrita no disco	129 MB/s	126 MB/s	135 MB/s
Relação com VMs ligadas		-2%	5%

Tabela 15 – Resultados da VM1 nos 3 cenários

À semelhança dos valores da VM1, os resultados para a VM2 também foram melhores quando os agentes de testes estavam desligados para ação de *undeployed*.

	VMs ligadas c/ testes	VMs em *poweroff*	VMs em *undeployed*
4096 MB System RAM			
Velocidade RAM	2562 MB/s	5224 MB/s	6852 MB/s
Relação com VMs ligadas		104%	167%

Hardware Tests			
Velocidade de escrita no disco	51 MB/s	56 MB/s	70 MB/s
Relação com VMs ligadas		10%	37%

Tabela 16 – Resultados da VM2 nos 3 cenários

Os resultados para a VM3 foram positivos para o cenário onde os agentes de testes estão desligados no estado *undeployed*.

	VMs ligadas c/ testes	**VMs em *poweroff***	**VMs em *undeployed***
4096 MB System RAM			
Velocidade RAM	4870 MB/s	5970 MB/s	6972 MB/s
Relação com VMs ligadas		23%	43%
Hardware Tests			
Velocidade de escrita no disco	23 MB/s	30 MB/s	31 MB/s
Relação com VMs ligadas		30%	35%

Tabela 17 – Resultados da VM3 nos 3 cenários

Deste modo confirma-se que o melhor cenário para a otimização dos recursos da *cloud* e para a elasticidade dos mesmos é quando os agentes de testes são ligados e desligados, e quando a ação de desligar toma a ação de *undeployed* e permite que os recursos dos anfitriões sejam libertados.